Stay in Your Lane:
Scalable Large-Integer Arithmetic in CKKS

Hyeongmin Choe, Robin Köstler, Tim Seuré

University of Luxembourg, Esch-sur-Alzette, Luxembourg
hyeongmin.choe@uni.lu, robin.koestler@uni.lu, tim.seure@uni.lu

Abstract. Recent work has shown that the approximate homomorphic encryption scheme CKKS can perform exact modular arithmetic with large moduli. This enables high-precision integer computation: given bounds on the outputs of an arithmetic circuit, we can recover the exact integer results from the modular outputs using any sufficiently large modulus. The particular choice of modulus does not affect correctness. Long computations require periodic bootstrapping, which refreshes ciphertexts so that further operations remain possible. However, in Gao and Zheng's construction (Crypto '26) and its extensions by Wei *et al.* and Liu *et al.* (both ePrint '26), converting between the representations used for arithmetic and bootstrapping, which is required at every refresh, becomes increasingly expensive as the integers grow.

We reduce this overhead by choosing message spaces that fit the representations already used inside standard CKKS bootstrapping. Building on GBFV (Geelen and Vercauteren; Eurocrypt '25) and its interpretation within CKKS (Kim; Asiacrypt '26), we organize encrypted data into independent polynomial "lanes" (namely, the factors of $(\mathbb{R}[X]/(X^n+1))^{N/n}$) that already appear as intermediate representations during bootstrapping (as intermediate stages of CoeffToSlot and SlotToCoeff). This avoids additional conversions specific to the message encoding. Supported message spaces include integers modulo β^n+1, where n is a power of two and $\beta \geqslant 2$ is a small integer, selected prime fields of up to 1 024 bits, and extension fields. For integer messages, we also give conversions in both directions between an arithmetic representation (for operations on whole integers) and a digit representation (for operations on their encrypted base-β digits).

We evaluate our construction and our port of Gao and Zheng in Lattigo on the same AWS Graviton5 hardware, using one thread at $N = 2^{16}$ and targeting 128-bit security. Our bootstrapping latency remains nearly constant for $8 \leqslant n \leqslant 2^{14}$ and stays below 8 seconds across all tested message spaces, including 24 577-bit moduli ($n = 4\,096$, $\beta = 64$). For $8 \leqslant n \leqslant 1\,024$, this gives a speedup of 2.1× to 6.8× over the port, comparing arithmetic modulo 2^n+1 with their arithmetic modulo 2^n. These gains carry over to sustained computation: including bootstrapping to support arbitrary evaluation depth, we estimate amortized multiplication speedups of 3.2× to 12.8× over the port for $8 \leqslant n \leqslant 512$. As one concrete example, arithmetic bootstrapping takes 6.44 seconds per ciphertext packing 16 integers modulo $2^{4\,096}+1$. In sustained computation, the amortized cost is 0.22 seconds per integer multiplication at over 4 000 bits of precision, including refreshing. Our arithmetic-to-digit conversion has per-call timings comparable to our implementation of Wei *et al.*'s conversion for $64 \leqslant n \leqslant 1\,024$ and $\beta = 2$, and our analysis indicates better scaling for larger integers.

Keywords: Bootstrapping · Discrete CKKS · Finite Rings · Fully Homomorphic Encryption · High-Precision Integer Arithmetic

1 Introduction

We review large-modulus arithmetic using CKKS, then position and outline our contribution.

1.1 General Overview

Homomorphic Encryption. *Fully homomorphic encryption* (FHE) enables a party to evaluate arbitrary circuits on encrypted inputs without learning their contents. Since Gentry's first construction [Gen09], schemes have adopted different ways of representing and processing encrypted data.

For instance, BGV and BFV [BGV12, Bra12, FV12] support exact arithmetic over finite plaintext rings, FHEW and TFHE [DM15, CGGI16] specialize in Boolean computations, and CKKS [CKKS17] was designed for approximate real and complex arithmetic.

CKKS: From Approximate to Exact. CKKS is based on RLWE [LPR10, SSTX09] over the integer polynomial ring $\mathbb{Z}[X]/\langle X^N+1\rangle$, where N is a power of two. For encoding, the isomorphism $\mathbb{R}[X]/\langle X^N+1\rangle \cong \mathbb{C}^{N/2}$ identifies elements of its real extension with vectors of $N/2$ complex slots. Scaling and rounding the polynomial coefficients then gives integer polynomials. Homomorphic addition and multiplication act componentwise, allowing parallel computation. Encoding, encryption, and evaluation introduce small errors, giving approximate results; *bootstrapping* restores the capacity for evaluation beyond the depth supported by the initial ciphertext parameters. Since its first construction [CHK+18], CKKS bootstrapping has benefited from faster linear transformations [HHC19], sparse-secret encapsulation [BTPH22], and EvalRound [KPK+22]. More recently, SPRU [CK25], SHIP [CHKS25], and PaCo [CS26] have shown that CKKS bootstrapping can also be performed by working in the exponents of roots of unity.

Sufficiently separated discrete messages remain unambiguously decodable despite small errors, and their known domain enables homomorphic error reduction. Thus, approximate intermediate computations can still yield exact discrete outputs. This underlies discrete CKKS, including error cleaning by Drucker *et al.* [DMPS24], bootstrapping for bits [BCKS24] and small integers [BKSS25], and finite-field arithmetic [SS26, KLN+26, JL26]. *Functional bootstrapping* combines refreshing with evaluating a prescribed function on a discrete domain [BKSS25, AKP25, AKPZ26], extending CKKS beyond approximate computation.

Large-Modulus Arithmetic. Storing a large integer in one CKKS slot requires a precision that grows with its bit length. Splitting the integer into digits lowers the precision per slot, but multiplication, modular reduction, and refreshing then act on several slots. Using Kim and Noh's modular reduction [KN25], Kim combines radix digits with functional bootstrapping [Kim25b], later accelerating multiplication modulo b^k for small b [Kim26]. Both reduce the digits with functional bootstrapping after multiplication, with a number of calls that grows with the number of digits.

Boneh and Kim [BK25] use nested residue number systems; Peikert *et al.* [PZZ26] introduce decomposed BFV (dBFV), supporting arbitrary integer moduli through digits across multiple BFV ciphertexts. CLPX, by Chen *et al.* [CLPX18], replaces BFV's integer plaintext modulus with $X-\beta$, giving leveled arithmetic modulo β^N+1 without refreshing. Geelen and Vercauteren's GBFV [GV25] generalizes to other polynomial plaintext moduli, supporting packed arithmetic modulo cyclotomic primes and efficient arithmetic bootstrapping. Kim [Kim25a] interprets and bootstraps these ciphertexts in CKKS. Gao and Zheng's GZ [GZ26] uses triangle encoding and tailored linear maps for arithmetic modulo powers of two. Per ciphertext, arithmetic bootstrapping takes two CKKS bootstrapping calls; arithmetic-to-Boolean conversion takes linearly many in the word size, or amortized $\mathcal{O}(1)$ with batching. Concurrent large-prime [WMGZ26] and arbitrary-modulus extensions [LMWW26] achieve bit or digit conversion with unamortized $\mathcal{O}(1)$ calls per ciphertext.

1.2 Our Construction

Motivation. Large-modulus arithmetic offers a direct route to high-precision integer computation on encrypted data. For an addition–multiplication circuit over $\mathbb{Z}$ whose outputs lie in $\{-B, -B+$

$1, \ldots, B\}$, evaluation modulo any $M > 2B$ allows exact recovery of the integer outputs by taking centered representatives. Intermediate values may wrap around without affecting correctness, since modular reduction is a ring homomorphism. Correctness therefore requires a sufficiently large modulus, but does not prescribe its value. Larger moduli necessarily require more ciphertext space per integer, reducing packing efficiency at a fixed RLWE dimension. Kim's digit-based constructions [Kim25b, Kim26] and GZ and its extensions [GZ26, WMGZ26, LMWW26] also incur additional computational costs: Kim's multiplication latency increases with the number of digits, while GZ and its extensions require more expensive representation conversions during refreshing. Our goal is to avoid these additional costs, allowing flexible modulus sizes with nearly unchanged arithmetic and refreshing latency per ciphertext.

Theoretical Overview. We perform exact arithmetic and refreshing in the *polynomial lanes* $\mathcal{S}_n = \mathbb{R}[X]/\langle X^n + 1\rangle$ native to CKKS bootstrapping, where $n \in [\![2, N]\!]$ is a power of two; the decomposition $\mathcal{S}_N \cong \mathcal{S}_n^{N/n}$ packs N/n independent lanes into one ciphertext. The case $n = N$ recovers the full-ring setting of GBFV and Kim's CKKS-based bootstrapping [GV25, Kim25a]. Following GBFV [GV25], we use the finite *message ring* $\mathcal{A}_n = \mathcal{R}_n/\langle t\rangle$, where $\mathcal{R}_n = \mathbb{Z}[X]/\langle X^n + 1\rangle$ is the integer subring of each lane and $t \in \mathbb{Z}[X]$ has degree less than n and is coprime to $X^n + 1$ in $\mathbb{Q}[X]$. The lane ring determines how we compute, while t determines the message space. For $t = X - \beta$ with an integer $\beta \geqslant 2$, evaluation at $X = \beta$ gives $\mathcal{A}_n \cong \mathbb{Z}_{\beta^n+1}$, enabling modular arithmetic without carry propagation. By the motivation above, this modulus is as suitable for exact integer computation as 2^n. Choosing β and n jointly lets us accommodate the output bound while keeping the modulus small and retaining a high packing density of N/n independent messages per ciphertext. Other choices of t yield products of modular integer rings and selected prime or extension fields (Table 2); for example, a degree-32 lane holds two $\mathbb{F}_{17^4}$ elements, giving 4 096 per ciphertext at $N = 2^{16}$.

After removing the scale, each lane has the form $P + tI + e$, where P is the chosen bounded message representative, I is an integer overflow polynomial, and e is the approximation error. Since tI preserves the message but amplifies error in later multiplications, lazy flattening (LazyFlatten, Algorithm 1; A2A-I in GZ) restores levels and bounds I without reducing e; arithmetic bootstrapping (ArithBoot, Algorithm 2; A2A in GZ) additionally reduces e and is therefore slower than lazy flattening. These refreshing steps reuse partial CoeffToSlot and SlotToCoeff [HHC19]: overflow control uses the degree-N coefficient representation, while error extraction exposes individual coefficients in slots. These polynomial lanes also appear in NeuJeans [JPK+24], PaCo [CS26], and Kim's integer arithmetic modulo b^k [Kim26]. Working directly in them avoids additional encoding-specific maps during refreshing (Figure 1), motivating our title, "Stay in Your Lane".

For $t = X - \beta$, we also describe conversion in both directions between the arithmetic representation and base-β digits in CKKS slots. Only n digits are needed: $a = \beta^n$ is represented by $(-1, 0, \ldots, 0)$, while all other residues use digits between 0 and $\beta - 1$.

Experimental Results. At $N = 2^{16}$, our arithmetic bootstrapping takes less than 8 seconds in every tested configuration in Lattigo [Lat24] on an AWS Graviton5 instance, using one thread. We measure 6.20 to 7.85 seconds for $8 \leqslant n \leqslant 2^{14}$, and the tested extension-field configurations closely match their modular-integer baselines. Lazy flattening becomes faster as n grows, since fewer linear-transformation stages connect the lanes to the degree-N ring $\mathcal{S}_N$. For instance, it takes 1.78 seconds at $n = 8$ and 0.13 seconds at $n = 2^{14}$. Amortized multiplication, including refreshing, also has nearly

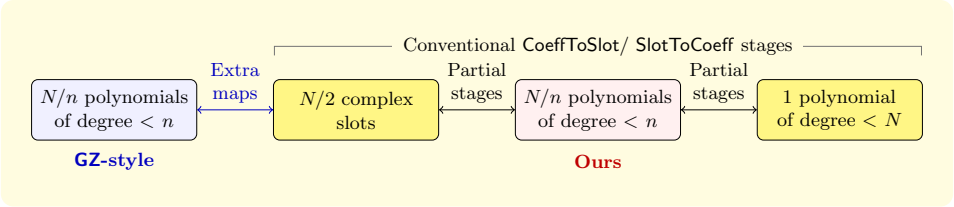


Fig. 1. Representation changes used in refreshing (schematic). Our arithmetic representation lies within the conventional CoeffToSlot/SlotToCoeff decomposition; GZ-style constructions require additional encoding-specific maps. Here, GZ-style refers to GZ [GZ26] and its arbitrary-modulus extensions [WMGZ26, LMWW26].

constant latency per ciphertext across the tested lane degrees. In particular, multiplying encrypted integers at 4 096-bit precision costs an estimated 0.217 seconds per product, amortized over 16 packed messages and including refreshing (Table 6). Together, these measurements support our goal: over the tested range, increasing n enlarges the modulus at fixed N and β without a corresponding increase in arithmetic or refreshing latency per ciphertext. Arithmetic-to-digit conversion for $\beta = 2$ takes 113 to 126 seconds per ciphertext for $64 \leqslant n \leqslant 1\,024$, using four functional bootstrapping calls. We will make the implementation publicly available upon publication.

Comparison to Related Work. Compared with CLPX [CLPX18], GBFV [GV25], and Kim's CKKS-based bootstrapping [Kim25a], we use degree-n plaintext lanes in place of full degree-N quotients, allowing us to choose n independently of the RLWE dimension. This directly generalizes Kim's interpretation of GBFV ciphertexts within CKKS [GV25, Kim25a].

Our arithmetic and refreshing resemble GZ [GZ26] and its concurrent extensions by Wei *et al.* [WMGZ26, Section 4.1] and Liu *et al.* [LMWW26, Section 3.2]. Using intermediate CKKS bootstrapping representations (Figure 1), we avoid their extra dense $(n/2) \times (n/2)$ maps, whose standard diagonal evaluation with baby-step giant-step costs $\mathcal{O}(n)$ plaintext–ciphertext multiplications and $\mathcal{O}(\sqrt{n})$ rotations per map. This comes at the cost of less message-space flexibility. Against our Lattigo port of published GZ on the same AWS hardware, lazy flattening (A2A-I in GZ) is 3.8× to 65× faster and arithmetic bootstrapping (A2A in GZ) is 2.1× to 6.8× faster for $8 \leqslant n \leqslant 1\,024$ (Figure 2). Both schemes pack N/n messages, modulo $2^n + 1$ (ours) or 2^n (GZ). Our refreshing uses 5 to 12 GiB; the port's memory grows linearly in n to 157 GiB at $n = 1\,024$, preventing comparisons at larger n. Including refreshing, we estimate amortized multiplication speedups of 3.2×, 6.2×, and 12.8× at $n = 8$, 64, and 512, respectively.

For fixed radix and input overflow bound, our arithmetic-to-digit conversion uses $\mathcal{O}(1)$ bootstrappings and $\mathcal{O}(\rho N^{1/\rho} + \log n)$ other ring operations; its inverse uses $\mathcal{O}(\rho N^{1/\rho})$ operations without bootstrapping. By comparison, Wei *et al.*'s arithmetic-to-Boolean conversion [WMGZ26] costs $\mathcal{O}(\rho N^{1/\rho} + n)$ operations beyond bootstrapping, while their reverse conversion costs $\mathcal{O}(n)$. Here ρ bounds the number of balanced groups per (partial) CoeffToSlot or SlotToCoeff transform and per half of our slot-permutation network (Section 7.3). Our arithmetic-to-digit conversion has comparable per-call timings (1.2× to 1.6× slower) to our implementation of Wei *et al.*'s conversion [WMGZ26] for $64 \leqslant n \leqslant 1\,024$, and is 2.5× to 3.2× faster per converted message. Since that implementation does not include all optimizations that Wei *et al.* describe, these ratios do not give a precise speedup over their construction. The complexity bounds suggest a growing advantage at larger n, where memory limits again prevent benchmarking the baseline.

Our finite-field support is more restrictive than that of Seuré and Suvanto [SS26], whose implementation supports a wider choice of fields $\mathbb{F}_{p^r}$ when both the characteristic p and extension degree r are small. In terms of time per field element, our implementation is competitive: it is 1.4× faster for $\mathbb{F}_{5^{16}}$, 1.8× faster for $\mathbb{F}_{37^{32}}$, and 1.5× slower for $\mathbb{F}_{17^4}$ (Table 5).

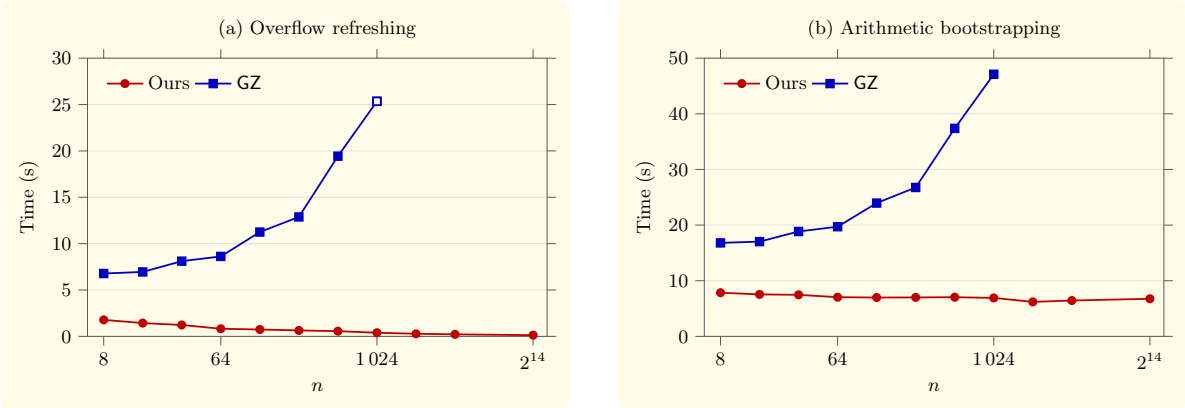


Fig. 2. Refreshing time per ciphertext: (a) lazy flattening versus GZ's A2A-I; (b) arithmetic bootstrapping versus A2A. Single-thread Lattigo on AWS Graviton5 at $N = 2^{16}$, packing N/n messages modulo $2^n + 1$ (ours) or 2^n (GZ). Points are ten-run means, except the open square (one run). Memory limits cap GZ at $n = 1\,024$ (157 GiB heap on a 256 GiB instance). See Section 8.1 and Table 4 for parameters and data.

1.3 Paper Organization

We introduce notation in Section 2 and review CKKS in Section 3. In Section 4 to Section 7, we develop the message rings, homomorphic arithmetic, refreshing procedures, and digit conversions. Section 8 presents our experimental evaluation. The supplementary material provides correctness bounds and further implementation details.

2 Notation

Basics. All logarithms are to base 2, unless explicitly stated otherwise. We write $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ for the integers, rationals, reals, and complex numbers, and $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ for the ring of integers modulo an integer $q \geqslant 1$. For a prime power q, $\mathbb{F}_q$ denotes the finite field with q elements. For integers $p \leqslant q$, set $[\![p, q]\!] = \{p, p+1, \ldots, q\}$ and $[\![n]\!] = [\![0, n-1]\!]$ for $n \geqslant 1$. We write $a \leftarrow A$ when a is uniformly sampled from a finite, nonempty set A. With imaginary unit i, we denote by $\zeta_n = \exp(2\pi\mathrm{i}/n)$ the primitive n-th root of unity with argument $2\pi/n$, for $n \geqslant 1$.

Modular Reduction and Rounding. For $x \in \mathbb{R}$ and an integer $q \geqslant 1$, $[x]_q \in (-q/2, q/2]$ is uniquely specified by $x - [x]_q \in q\mathbb{Z}$. Rounding to the nearest integer (ties downwards) is $\lfloor x \rceil = x - [x]_1$. Both act entrywise on tuples and coefficientwise on polynomials. In a ring R with $a, a', b \in R$, we write $a \bmod b$ for the image of a in $R/\langle b \rangle$, and $a \equiv_b a'$ means $a \bmod b = a' \bmod b$ in $R/\langle b \rangle$.

Vectors and Matrices. Vectors are columns, even when written horizontally with commas; $\mathbf{0}$ and $\mathbf{1}$ are the zero and all-one vectors of implicit length. For a vector $\mathbf{v}$ of length n, we write $\mathbf{v} = (v_i)_{i \in [\![n]\!]}$,

where v_i is its entry at index i, starting at 0. Maps extend entrywise: $f(\mathbf{a}) = (f(a_i))_{i\in[\![n]\!]}$ for $f : A \to B$ and $f(\mathbf{a}, \mathbf{b}) = (f(a_i, b_i))_{i\in[\![n]\!]}$ for $f : A \times B \to C$, and similarly for more arguments. For a ring R, we write $\mathbf{v} \odot \mathbf{w} = (v_i w_i)_{i\in[\![n]\!]}$ for the *Hadamard product* between $\mathbf{v}, \mathbf{w} \in R^n$. For $\mathbf{z} \in \mathbb{C}^n$ and $j \in \mathbb{Z}$, the *cyclic rotation* is $\mathsf{Rot}_j(\mathbf{z}) = (z_{i+j})_{i\in[\![n]\!]}$, with indices modulo n. We write $\mathbf{I}_m$ for the $m \times m$ identity matrix, $\mathsf{diag}(\mathbf{v}) \in R^{n\times n}$ for the diagonal matrix with diagonal $\mathbf{v} \in R^n$, and $\mathbf{M}^* \in \mathbb{C}^{n\times m}$ for the conjugate transpose of $\mathbf{M} \in \mathbb{C}^{m\times n}$. For an $n \times n$ matrix $\mathbf{M}$, we write $\mathbf{I}_m \otimes \mathbf{M}$ for the block-diagonal matrix of size $(mn) \times (mn)$, having m copies of $\mathbf{M}$ on its diagonal.

Norms. For $\mathbf{z} \in \mathbb{C}^n$, we use $\|\mathbf{z}\|_1 = \sum_{i\in[\![n]\!]} |z_i|$ and $\|\mathbf{z}\|_\infty = \max_{i\in[\![n]\!]} |z_i|$. Polynomial norms use coefficient vectors: $P = \sum_{i\in[\![n]\!]} P_i X^i$ has $\|P\|_1 = \sum_{i\in[\![n]\!]} |P_i|$ and $\|P\|_\infty = \max_{i\in[\![n]\!]} |P_i|$. For tuples, we concatenate the corresponding vectors.

3 Background on CKKS

We recall the CKKS interfaces used below [CKKS17, CHK+18]. The scheme uses RLWE encryption [LPR10, SSTX09]; see [CKKS17, LM21, LMSWS22] for security notions and analyses.

Encoding and Decoding. For a power of two $d \geqslant 2$, let the *evaluation map* $\mathsf{eval}_d : \mathcal{S}_d \to \mathbb{C}^{d/2}$ be given by $\mathsf{eval}_d(P) = (P(\zeta_{2d}^{5^i}))_{i\in[\![d/2]\!]}$. At an integer scale $\Delta \gg 0$, we encode $N/2$ complex slots by $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{z}) = \lfloor \Delta \mathsf{eval}_N^{-1}(\mathbf{z}) \rceil$ and decode by $\mathsf{Dcd}_{\mathbb{C}}(m) = \mathsf{eval}_N(m/\Delta)$, leaving the scale implicit.

Encryption and Decryption. We use the simplified *modulus chain* $q_\ell = q_0 \Delta^\ell$ for $\ell \in [\![0, L]\!]$; [CHK+19] describes the RNS implementation. Write $\mathcal{R}_{N,q}$ for the elements of $\mathcal{R}_N$ with coefficients in $(-q/2, q/2]$. A *level-ℓ ciphertext* is a tuple $\mathbf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1) \in \mathcal{R}_{N,q_\ell}^2$, decrypted as $\mathsf{Dec}_{s,\ell}(\mathbf{ct}) = [\mathsf{ct}_0 s + \mathsf{ct}_1]_{q_\ell}$ under secret key $s \in \mathcal{R}_N$. For $m \in \mathcal{R}_N$, fresh encryption $\mathbf{ct} \leftarrow \mathsf{Enc}_s(m)$ at level L satisfies $\mathsf{Dec}_{s,L}(\mathbf{ct}) = [m + e_{\mathsf{enc}}]_{q_L}$, where $e_{\mathsf{enc}} \in \mathcal{R}_N$ is sampled from the chosen RLWE error distribution. We typically write the decrypted polynomial as $\Delta P + e$, where $P \in \mathcal{S}_N$ is the ideal message and $e \in \mathcal{S}_N$ is the error. We call $\mathbf{z} = \mathsf{eval}_N(P) \in \mathbb{C}^{N/2}$ the ideal *slot vector* of $\mathbf{ct}$, and its entries the *slot values*. Thus, $P = \mathsf{eval}_N^{-1}(\mathbf{z})$, with rounded encoding $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{z}) = \lfloor \Delta P \rceil$.

Basic Homomorphic Operations. Let $\mathbf{ct}, \mathbf{ct}' \in \mathcal{R}_{N,q_\ell}^2$ be level-ℓ ciphertexts under secret key s, with $\mathsf{Dec}_{s,\ell}(\mathbf{ct}) = \Delta P + e$ and $\mathsf{Dec}_{s,\ell}(\mathbf{ct}') = \Delta P' + e'$, for $P, P', e, e' \in \mathcal{S}_N$. Let $m \in \mathcal{R}_N$ be a plaintext polynomial. Each call in Table 1 returns a ciphertext $\mathbf{ct}_{\mathsf{out}} \in \mathcal{R}_{N,q_{\ell'}}^2$. The displayed decryption relations assume no wraparound at the output modulus; errors are measured before division by the scale. Here e_{ks} and e_{rs} denote the errors introduced by key switching and rescaling in each call. KeySwitch uses the key-switching key $\mathsf{swk}_{s'\leftarrow s}$, and ciphertext–ciphertext Mul includes relinearization with $\mathsf{swk}_{s\leftarrow s^2}$. Multiplication of two scale-Δ ciphertexts gives scale Δ^2 and is typically followed by Rescale to restore scale Δ. For $\ell \geqslant 1$, Rescale divides the scale by Δ and consumes one level. ModRaise requires $\ell' \geqslant \ell$ and introduces an integer overflow $I \in \mathcal{R}_N$. The automorphisms $\tau : X \mapsto X^{-1}$ and $\sigma_j : X \mapsto X^{5^j}$ induce slot conjugation and rotation: for $P = \mathsf{eval}_N^{-1}(\mathbf{z})$, $\mathsf{eval}_N(\tau(P)) = \overline{\mathbf{z}}$ and $\mathsf{eval}_N(\sigma_j(P)) = \mathsf{Rot}_j(\mathbf{z})$. For $j \in (N/2)\mathbb{Z}$, rotation returns the input without key switching or added error.

Table 1. Basic CKKS operations: output secret key, level, and decryption.

Call producing $\mathbf{ct}_{\mathsf{out}}$	Secret key $\tilde{s}$	Level $\tilde{\ell}$	$\mathsf{Dec}_{\tilde{s},\tilde{\ell}}(\mathbf{ct}_{\mathsf{out}})$
$\mathsf{Add}(m, \mathbf{ct})$	s	ℓ	$m + \Delta P + e$
$\mathsf{Add}(\mathbf{ct}, \mathbf{ct}')$	s	ℓ	$\Delta(P + P') + e + e'$
$\mathsf{Mul}(m, \mathbf{ct})$	s	ℓ	$\Delta mP + me$
$\mathsf{Mul}(\mathbf{ct}, \mathbf{ct}')$	s	ℓ	$\Delta^2 PP' + \Delta(Pe' + P'e) + ee' + e_{\mathsf{ks}}$
$\mathsf{KeySwitch}_{s' \leftarrow s}(\mathbf{ct})$	s'	ℓ	$\Delta P + e + e_{\mathsf{ks}}$
$\mathsf{Conj}(\mathbf{ct})$	s	ℓ	$\Delta\tau(P) + \tau(e) + e_{\mathsf{ks}}$
$\mathsf{Rot}_j(\mathbf{ct})$	s	ℓ	$\Delta\sigma_j(P) + \sigma_j(e) + e_{\mathsf{ks}}$
$\mathsf{Rescale}(\mathbf{ct})$	s	$\ell - 1$	$P + e/\Delta + e_{\mathsf{rs}}$
$\mathsf{ModRaise}_{\ell' \leftarrow \ell}(\mathbf{ct})$	s	ℓ'	$\Delta P + e + q_\ell I$

Homomorphic Linear Transformations. Let $\mathbf{ct}$ be a level-ℓ ciphertext with slot vector $\mathbf{z}$ and let $\mathbf{M} \in \mathbb{C}^{(N/2)\times(N/2)}$. Define its *cyclic diagonals* by $\mathsf{Diag}_j(\mathbf{M}) = (M_{i,i+j})_{i\in[\![N/2]\!]}$, with column indices modulo $N/2$. The *diagonal method* [HS14] uses the identity $\mathbf{Mz} = \sum_{j\in[\![N/2]\!]} \mathsf{Diag}_j(\mathbf{M}) \odot \mathsf{Rot}_j(\mathbf{z})$. Evaluating this sum with Rot, Mul, and Add, followed by Rescale, gives a direct implementation of $\mathsf{LinTrans}(\mathsf{Ecd}_{\mathbb{C}}(\mathbf{M}), \mathbf{ct})$, where $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{M})$ denotes the encoded nonzero diagonals. Its output has slot vector $\mathbf{Mz}$ at level $\ell-1$. Some matrices factor into products of *diagonally sparse* matrices, each having few nonzero cyclic diagonals, reducing operation counts at the cost of additional levels [HHC19]. For such implementations, $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{M})$ denotes the encoded diagonals of the factors. If the nonzero diagonal indices of a matrix lie in an arithmetic progression of length D modulo $N/2$, baby-step giant-step evaluation uses $\mathcal{O}(D)$ plaintext–ciphertext multiplications, $\mathcal{O}(\sqrt{D})$ rotations, and one level [HS14]. LinTrans leaves the implementation choice implicit; its error includes propagated input error and encoding, key-switching, and rescaling errors [KPK+22].

Partial CoeffToSlot and SlotToCoeff. Fix powers of two $d, d' \in [\![2, N]\!]$. Applying $\mathsf{eval}_d : \mathcal{S}_d \to \mathbb{C}^{d/2}$ to each of the N/d polynomial components gives a ring isomorphism $\mathsf{eval}_d : \mathcal{S}_d^{N/d} \to \mathbb{C}^{N/2}$, denoted by the same symbol. Each polynomial tuple thus corresponds to a unique slot vector. The induced *packing isomorphism* $\Phi_{d'\leftarrow d} : \mathcal{S}_d^{N/d} \to \mathcal{S}_{d'}^{N/d'}$, given by $\Phi_{d'\leftarrow d} = \mathsf{eval}_{d'}^{-1} \circ \mathsf{eval}_d$, preserves slots: $\mathbf{P}$ and $\Phi_{d'\leftarrow d}(\mathbf{P})$ have the same image in $\mathbb{C}^{N/2}$.

Given $\mathbf{P} \in \mathcal{S}_d^{N/d}$, we want to regroup its $(N/d) \cdot d = N$ coefficients into N/d' polynomials of degree less than d', without changing their values. For this, we transform the slot vector $\mathbf{z} = \mathsf{eval}_d(\mathbf{P})$ so that its new interpretation in $\mathcal{S}_{d'}^{N/d'}$ contains the N coefficients of $\mathbf{P}$, but in a rearranged order. The matrix $\mathbf{U}_{d'\leftarrow d}$ below achieves this: the new polynomial tuple is $\mathsf{eval}_{d'}^{-1}(\mathbf{U}_{d'\leftarrow d}\mathbf{z})$. For $d \leqslant d'$, set $\mathbf{U}_{d'\leftarrow d} = \mathbf{U}_{d'\leftarrow d'/2} \cdots \mathbf{U}_{2d\leftarrow d}$, with the following elementary factors:

$$\mathbf{U}_{2k\leftarrow k} = \mathbf{I}_{N/(2k)} \otimes \begin{pmatrix} \mathbf{I}_{k/2} & \mathbf{W}_{k/2} \\ \mathbf{I}_{k/2} & -\mathbf{W}_{k/2} \end{pmatrix}, \qquad \mathbf{W}_{k/2} = \mathsf{diag}\big((\zeta_{4k}^{5^i})_{i\in[\![k/2]\!]}\big),$$

where $k \in [\![2, N/2]\!]$ is a power of two. For $d > d'$, set $\mathbf{U}_{d'\leftarrow d} = \mathbf{U}_{d\leftarrow d'}^{-1} = (d'/d)\mathbf{U}_{d\leftarrow d'}^{*}$. These give *partial SlotToCoeff* when increasing the degree and *partial CoeffToSlot* when decreasing it [HHC19, JPK+24, CS26]. The induced *coefficient map* $\kappa_{d'\leftarrow d} : \mathcal{S}_d^{N/d} \to \mathcal{S}_{d'}^{N/d'}$, given by $\kappa_{d'\leftarrow d} = \mathsf{eval}_{d'}^{-1} \circ \mathbf{U}_{d'\leftarrow d} \circ \mathsf{eval}_d$, simply transports coefficients from $\mathcal{S}_d^{N/d}$ to $\mathcal{S}_{d'}^{N/d'}$, reordering them without changing their values. For $d < d'$ and $\mathbf{P} = (P_i)_{i\in[\![N/d]\!]} \in \mathcal{S}_d^{N/d}$, each group of d'/d adjacent input polynomials is packed into one output polynomial. Their coefficients are interleaved: consecutive

coefficients of each input polynomial lie d'/d positions apart and remain in their natural order. The input polynomials themselves occur in bit-reversed order within this interleaving; for example, four polynomials interleave as P_0, P_2, P_1, P_3.

For homomorphic evaluation, if $d < d'$, then grouping the $\log(d'/d)$ sparse factors and using baby-step giant-step evaluates either direction in $\rho \in [\![1, \log(d'/d)]\!]$ levels with $\mathcal{O}\big(\rho(d'/d)^{1/\rho}\big)$ plaintext–ciphertext multiplications and $\mathcal{O}\big(\rho(d'/d)^{1/(2\rho)}\big)$ rotations [HS14, HHC19].

4 Message Rings and Representatives

We define bounded polynomial representatives and give concrete message rings.

4.1 Lifting and Projecting

Fix a power of two $n \in [\![2, N]\!]$ and $t \in \mathbb{Z}[X]$ of degree less than n, coprime to $X^n + 1$ in $\mathbb{Q}[X]$. Set $\mathcal{A}_n = \mathcal{R}_n/\langle t \rangle$ and let $\mathsf{Proj}_n : \mathcal{R}_n \twoheadrightarrow \mathcal{A}_n$ be the *quotient projection*. We identify t with its images in $\mathcal{R}_n$ and $\mathcal{S}_n$; all divisions by t take place in $\mathcal{S}_n$, where it is invertible. The degree restriction loses no generality, since t can be reduced modulo $X^n + 1$. The maps below apply componentwise to the full message space $\mathcal{A}_n^{N/n}$. GBFV's *flattening map* [GV25, Section 3.1] generalizes the hat encoder of CLPX [CLPX18, Section 4.1]. In our notation, this map $\mathsf{Flatten}_n : \mathcal{R}_n \to \mathcal{R}_n$ and the induced *lifting map* $\mathsf{Lift}_n : \mathcal{A}_n \to \mathcal{R}_n$ are given by:

$$\mathsf{Flatten}_n(P) = t\left[\frac{P}{t}\right]_1, \qquad \mathsf{Lift}_n(a) = \mathsf{Flatten}_n(P) \quad \text{for any } \mathsf{Proj}_n(P) = a.$$

Flattening selects a canonical integer representative of each residue class, independently of the choice of P. Thus $\mathsf{Flatten}_n \circ \mathsf{Flatten}_n = \mathsf{Flatten}_n$ and $\mathsf{Proj}_n \circ \mathsf{Lift}_n = \mathrm{id}_{\mathcal{A}_n}$. Since $\mathsf{Lift}_n(a)/t = [P/t]_1$ and $\|tQ\|_\infty \leqslant \|t\|_1 \|Q\|_\infty$ for $Q \in \mathcal{S}_n$, every $a \in \mathcal{A}_n$ satisfies:

$$\left\| \frac{\mathsf{Lift}_n(a)}{t} \right\|_\infty \leqslant \frac{1}{2}, \qquad \|\mathsf{Lift}_n(a)\|_\infty \leqslant \frac{\|t\|_1}{2}. \tag{1}$$

Every representative P of a has the form $P = \mathsf{Lift}_n(a) + tI$, where $I \in \mathcal{R}_n$ is the *overflow polynomial* [LMWW26]. Since Proj_n is a ring homomorphism with kernel $\langle t \rangle$, representatives P, P' of a, a' satisfy $P + P' = \mathsf{Lift}_n(a + a') + tI_{\mathsf{add}}$ and $PP' = \mathsf{Lift}_n(aa') + tI_{\mathsf{mul}}$ for some $I_{\mathsf{add}}, I_{\mathsf{mul}} \in \mathcal{R}_n$.

4.2 Concrete Message Spaces

Choices of t yield modular integer rings, finite fields, and their products as lane message spaces. The following result packs r modular integers per lane, hence rN/n per ciphertext; its linear case $r = 1$ uses the CLPX construction [CLPX18]:

Proposition 1 (Residue-ring decomposition). *Let $t = X^r - \beta^r$, where $r \in [\![1, n/2]\!]$ is a power of two and $\beta \geqslant 2$ is an integer. If $r = 1$ or if β is even, we have an isomorphism of rings $\mathcal{A}_n \cong \mathbb{Z}_M^r$ for $M = \beta^n + 1$.*

Proof. Reducing $X^n + 1$ modulo t and substituting $X = \beta Y$ gives $\mathcal{A}_n \cong \mathbb{Z}_M[Y]/\langle Y^r - 1 \rangle$, since β is invertible modulo M. The case $r = 1$ is immediate. Otherwise, M is odd and $\omega = \beta^{2n/r}$ satisfies $\omega^{r/2} \equiv_M -1$. The Fourier decomposition over rings [Für09, Section 2] therefore gives $\mathbb{Z}_M[Y]/\langle Y^r - 1 \rangle \cong \mathbb{Z}_M^r$. □

For prime M, GBFV's decomposition [GV25, Lemma 9 and Equation (4)] with $m = 2n$, $k = r$, and $b = c$ gives the following result for arbitrary c, recovering Proposition 1 when $c = \beta^r$:

Proposition 2 (Finite-field decomposition). *Let $t = X^r - c$, where $r \in [\![1, n/2]\!]$ is a power of two and $c \geqslant 2$ is an integer. If $M = c^{n/r} + 1$ is prime, then $\mathcal{A}_n \cong \mathbb{F}_{M^u}^{r/u}$, where $u = \mathrm{ord}_{2n}(M)$.*

For non-binomial choices of t, [GV25, Lemmas 2 and 3] give the following prime-field criterion:

Proposition 3 (Prime-field quotients). *Let $t \in \mathbb{Z}[X]$ have degree less than n, and define $p = \left|\prod_{j=0}^{n-1} t(\zeta_{2n}^{2j+1})\right| \in \mathbb{Z}$. If p is prime, then $\mathcal{A}_n \cong \mathbb{F}_p$.*

Table 2 lists twelve choices at $N = 2^{16}$: two composite moduli at different packing densities, four extension fields, and four large prime fields. It gives their packing capacities and the norms $\|t\|_1$ and $\|1/t\|_1$. The former bounds representative size (1); keeping the latter small limits error growth in the refreshing steps described in Section 6.

Table 2. Message spaces and packing capacities for $N = 2^{16}$, with N/n lanes per ciphertext. Each prime p_k has k bits and is determined by Proposition 3 for its row. Norms use representatives of degree less than n in $\mathcal{S}_n$; inverse norms are rounded to three decimal places.

$\log n$	t	$\|t\|_1$	$\|1/t\|_1$	Ring / field	Elements / ciphertext
6	$X - 2$	3	1.000	$\mathbb{Z}_{2^{64}+1}$	1 024
6	$X^2 - 4$	5	0.333	$\mathbb{Z}_{2^{64}+1}$	2 048
10	$X - 2$	3	1.000	$\mathbb{Z}_{2^{1\,024}+1}$	64
10	$X^2 - 4$	5	0.333	$\mathbb{Z}_{2^{1\,024}+1}$	128
5	$X^{16} - 2$	3	0.600	$\mathbb{F}_{5^{16}}$	2 048
5	$X^8 - 2$	3	0.882	$\mathbb{F}_{17^4}$	4 096
6	$X^{32} - 6$	7	0.189	$\mathbb{F}_{37^{32}}$	1 024
10	$X^{128} - 2$	3	0.992	$\mathbb{F}_{257^8}$	1 024
6	$X^4 - X^3 + X + 4$	7	0.746	$\mathbb{F}_{p_{128}}$	1 024
7	$X^6 + X^2 + X + 4$	7	0.610	$\mathbb{F}_{p_{256}}$	512
8	$X^7 + X^6 + X + 4$	7	0.527	$\mathbb{F}_{p_{512}}$	256
9	$X^6 + X^5 + X^3 + 4$	7	0.627	$\mathbb{F}_{p_{1\,024}}$	128

5 Our Framework within CKKS

We fix t and the message ring $\mathcal{A}_n = \mathcal{R}_n/\langle t \rangle$ as in Section 4.1, with one message in each of N/n polynomial lanes. We extend the lifting map componentwise, writing $\mathsf{Lift}_n : \mathcal{A}_n^{N/n} \to \mathcal{R}_n^{N/n}$. The packing isomorphisms $\Phi_{N \leftarrow n}$ and $\Phi_{n \leftarrow N}$ from Section 3 connect these lanes to the CKKS plaintext ring.

5.1 Ciphertexts and Decryption

At scale Δ, our *encoding map* $\mathsf{Ecd}_{\mathcal{A}_n} : \mathcal{A}_n^{N/n} \to \mathcal{S}_N$ is given by $\mathsf{Ecd}_{\mathcal{A}_n}(\mathbf{a}) = \Delta \Phi_{N \leftarrow n}(\mathsf{Lift}_n(\mathbf{a}))$. We round its coefficients for encryption and plaintext operands. A ciphertext $\mathbf{ct} \in \mathcal{R}_{N,q_\ell}^2$ under secret key s represents $\mathbf{a} \in \mathcal{A}_n^{N/n}$ at scale Δ if it satisfies $\mathsf{Dec}_{s,\ell}(\mathbf{ct}) = \mathsf{Ecd}_{\mathcal{A}_n}(\mathbf{a}) + \Phi_{N \leftarrow n}(\Delta t \mathbf{I} + \mathbf{e})$, where $\mathbf{I} \in \mathcal{R}_n^{N/n}$ is the overflow and $\mathbf{e} \in \mathcal{S}_n^{N/n}$ is the total approximation error, including rounding.

Unpacking gives $\Phi_{n\leftarrow N}(\mathsf{Dec}_{s,\ell}(\mathbf{ct})) = \Delta\mathsf{Lift}_n(\mathbf{a}) + \Delta t\mathbf{I} + \mathbf{e}$. If $\|\mathbf{e}\|_\infty < \Delta/2$, we recover $\mathbf{a}$ from $P = \mathsf{Dec}_{s,\ell}(\mathbf{ct})$ by computing:

$$\mathsf{Dcd}_{\mathcal{A}_n}(P) = \mathsf{Proj}_n\left(\left\lfloor \frac{\Phi_{n\leftarrow N}(P)}{\Delta} \right\rceil\right). \tag{2}$$

Division by Δ and rounding yield $\mathsf{Lift}_n(\mathbf{a}) + t\mathbf{I}$, and projection removes $t\mathbf{I}$. Figure 3 summarizes these maps.

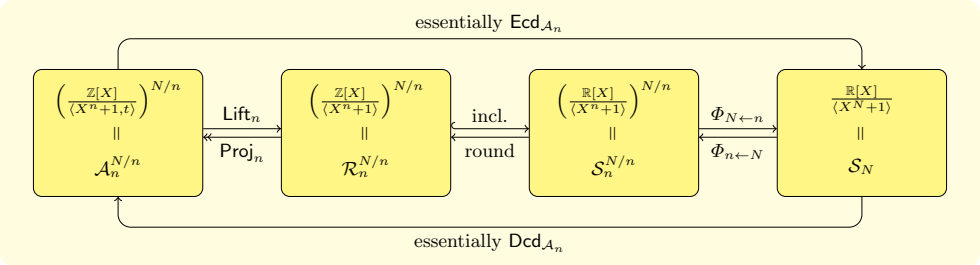


Fig. 3. Maps between the message space and the CKKS plaintext ring. The encoding map $\mathsf{Ecd}_{\mathcal{A}_n}$ additionally scales by Δ; the decoding map $\mathsf{Dcd}_{\mathcal{A}_n}$ divides by Δ before coefficientwise rounding.

Remark 1. Correct decoding after fresh encryption and homomorphic CKKS operations relies on $\|\mathbf{e}\|_\infty < \Delta/2$ and controlled overflow $\mathbf{I}$ to avoid wraparound modulo q_ℓ. Supplementary Material A gives worst-case sufficient conditions (Proposition 4). We defer these technical bounds for clarity; their conservatism makes heuristic estimates of overflow and error growth, supported by experiments, more useful for practical parameter selection.

5.2 Encryption and Homomorphic Operations

Fresh encryption of $\mathbf{a} \in \mathcal{A}_n^{N/n}$ is $\mathbf{ct} \leftarrow \mathsf{Enc}_s\big(\lfloor \mathsf{Ecd}_{\mathcal{A}_n}(\mathbf{a})\rceil\big)$. The ciphertext has level L and zero overflow; its total error comes from rounding and encryption.

Further, since packing and projection preserve addition and multiplication, standard CKKS operations perform arithmetic independently in each lane. Table 3 gives their effect on the decoded messages, provided the output has no modulus wraparound and its total lane error is below $\Delta/2$. Here the ciphertexts $\mathbf{ct}, \mathbf{ct}'$ represent $\mathbf{a}, \mathbf{a}'$ at level ℓ and scale Δ under the same secret key s.

Table 3. Arithmetic in the lanes. Multiplication is followed by rescaling to restore scale Δ and requires $\ell \geqslant 1$.

Call producing $\mathbf{ct}_{\mathsf{out}}$	Output level	Decoded message
$\mathsf{Add}(\lfloor \mathsf{Ecd}_{\mathcal{A}_n}(\mathbf{a}')\rceil, \mathbf{ct})$	ℓ	$\mathbf{a} + \mathbf{a}'$
$\mathsf{Add}(\mathbf{ct}, \mathbf{ct}')$	ℓ	$\mathbf{a} + \mathbf{a}'$
$\mathsf{Rescale}(\mathsf{Mul}(\lfloor \mathsf{Ecd}_{\mathcal{A}_n}(\mathbf{a}')\rceil, \mathbf{ct}))$	$\ell - 1$	$\mathbf{a} \odot \mathbf{a}'$
$\mathsf{Rescale}(\mathsf{Mul}(\mathbf{ct}, \mathbf{ct}'))$	$\ell - 1$	$\mathbf{a} \odot \mathbf{a}'$

Addition adds the total input errors; multiplication amplifies them according to representative size. Plaintext encoding, ciphertext–ciphertext relinearization, and rescaling contribute their respective errors. All arithmetic operations can increase the overflow, which enlarges representatives and amplifies errors in later multiplications. This motivates the refreshing procedures in Section 6.

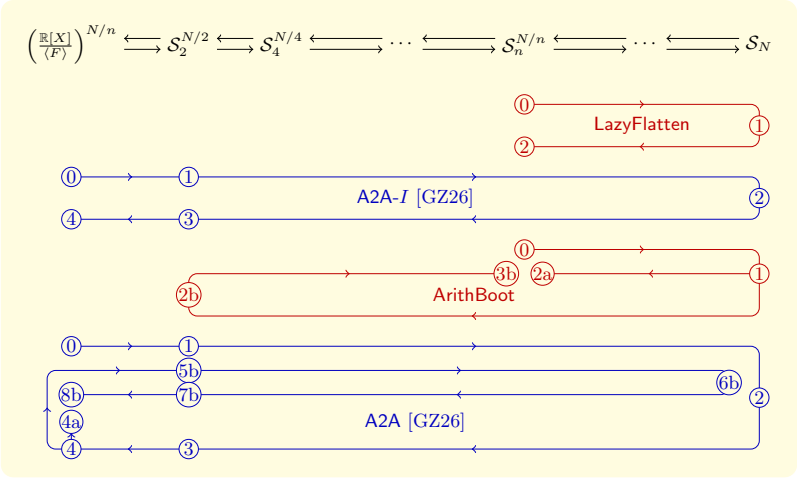


Fig. 4. Refreshing paths: ours (red) and GZ (blue), with $F = X^n - X + 2$. Lazy flattening refreshes overflow via $\mathcal{S}_N$; arithmetic bootstrapping also visits the complex slots to extract the error. Our asymptotic speedup comes from avoiding the far-left quotient ring and its dense encoding-specific maps, which require $\mathcal{O}(n)$ plaintext–ciphertext multiplications. Arithmetic bootstrapping removes the input error by subtracting 3b from 2a. GZ's A2A performs overflow refreshing A2A-I (0 to 4), followed by error extraction A2A-e (4 to 8b), then subtracts 8b from 4a [GZ26, Section 5.1]. The EvalRound-based variant [LMWW26, Section 4.3] fuses both loops into one round trip through $\mathcal{S}_N$, but retains these dense maps and requires accuracy over a larger integer range for EvalRound.

6 Ciphertext Refreshing

Similarly to [GZ26], we refresh ciphertexts representing $\mathbf{a} \in \mathcal{A}_n^{N/n}$ with two algorithms. Both take and return ciphertexts under secret key s at scale Δ. Lazy flattening restores levels and bounds the overflow $\mathbf{I}$ without reducing the input error $\mathbf{e}$; arithmetic bootstrapping additionally refreshes $\mathbf{e}$. We omit error refreshing alone, since skipping overflow refreshing saves little time.

As shown in Figure 4, arithmetic bootstrapping branches in $\mathcal{S}_N$: one ciphertext returns directly to $\mathcal{S}_n^{N/n}$, while the other extracts the error through $\mathcal{S}_2^{N/2} \cong \mathbb{C}^{N/2}$. Subtracting the latter from the former in the lanes cancels the input error.

In Algorithm 1 and Algorithm 2, the gray annotation of each step gives, from left to right, its plaintext representation and its output ciphertext level.

6.1 Lazy Flattening

Following [GZ26], lazy flattening divides each lane by t, collects coefficients in $\mathcal{S}_N$, and changes scale from Δ to q_0. The overflow term $q_0\kappa_{N\leftarrow n}(\mathbf{I})$ then vanishes modulo q_0, since $\kappa_{N\leftarrow n}(\mathbf{I}) \in \mathcal{R}_N$. Modulus raising under an ephemeral ternary secret key $s' \in \mathcal{R}_N$ of small Hamming weight h' restores levels with controlled new overflow $\tilde{\mathbf{I}}$ (sparse-secret encapsulation [BTPH22]). Reversing the transformation restores the lanes and scale Δ, retaining the input error $\mathbf{e}$ with added evaluation error. We use key-switching keys from s to s' at level 0 and back at L, and precompute encodings of:

$$\mathbf{V}_{N\leftarrow n} = \frac{q_0}{\Delta} \cdot \mathbf{U}_{N\leftarrow n} \cdot \mathbf{D}_t^{-1}, \qquad \mathbf{V}_{n\leftarrow N} = \frac{\Delta}{q_0} \cdot \mathbf{D}_t \cdot \mathbf{U}_{n\leftarrow N},$$

where $\mathbf{D}_t = \mathbf{I}_{N/n} \otimes \mathsf{diag}(\mathsf{eval}_n(t)) \in \mathbb{C}^{(N/2)\times(N/2)}$ implements lane multiplication by t. The diagonal factors can be absorbed into the sparse partial transformations. Their level costs are $\ell_{N\leftarrow n}$ and

$\ell_{n\leftarrow N}$, with $L \geqslant \ell_{n\leftarrow N}$. Set $\ell = \ell_{N\leftarrow n}$ and $\tilde{\ell} = L - \ell_{n\leftarrow N}$. If the input and evaluation errors are small enough, the overflow satisfies $\|\tilde{\mathbf{I}}\|_\infty \leqslant h'/2 + 1$. See Supplementary Material A.5 for the condition and exact bounds.

Algorithm 1: LazyFlatten

Input: $\mathbf{ct}_{\mathsf{in}}$ representing $\mathbf{a} \in \mathcal{A}_n^{N/n}$ at level ℓ.
Output: $\mathbf{ct}_{\mathsf{out}}$ representing $\mathbf{a}$ at level $\tilde{\ell}$, with refreshed overflow.

1: $\mathbf{ct} \leftarrow \mathsf{LinTrans}(\mathsf{Ecd}_{\mathbb{C}}(\mathbf{V}_{N\leftarrow n}), \mathbf{ct}_{\mathsf{in}})$	$\mathcal{S}_N$	0
2: $\mathbf{ct} \leftarrow \mathsf{KeySwitch}_{s'\leftarrow s}(\mathbf{ct})$	$\mathcal{S}_N$	0
3: $\mathbf{ct} \leftarrow \mathsf{ModRaise}_{L\leftarrow 0}(\mathbf{ct})$	$\mathcal{S}_N$	L
4: $\mathbf{ct} \leftarrow \mathsf{KeySwitch}_{s\leftarrow s'}(\mathbf{ct})$	$\mathcal{S}_N$	L
5: $\mathbf{ct}_{\mathsf{out}} \leftarrow \mathsf{LinTrans}(\mathsf{Ecd}_{\mathbb{C}}(\mathbf{V}_{n\leftarrow N}), \mathbf{ct})$	$\mathcal{S}_n^{N/n}$	$\tilde{\ell}$
6: **return** $\mathbf{ct}_{\mathsf{out}}$		

6.2 Arithmetic Bootstrapping

Arithmetic bootstrapping follows lazy flattening through modulus raising, then multiplies by $t(X^{N/n})$ in $\mathcal{S}_N$, making the message and refreshed overflow multiples of q_0. We retain this ciphertext and reduce a copy modulo q_0 to isolate the error. After a second modulus raising, this copy is transformed into complex slots, where EvalMod [CHK+18] approximately removes the fresh integer overflow by reduction modulo one in each real and imaginary component. Both branches return to the lanes; subtracting the error branch from the retained ciphertext cancels the input error. We reuse $\mathbf{V}_{N\leftarrow n}$ and the ephemeral secret key s' from lazy flattening, and precompute encodings of:

$$\mathbf{W}_{2\leftarrow N} = \frac{\Delta}{q_0} \cdot \mathbf{U}_{2\leftarrow N}, \qquad \mathbf{W}_{n\leftarrow N} = \frac{\Delta}{q_0} \cdot \mathbf{U}_{n\leftarrow N}.$$

The maps $\mathbf{W}_{2\leftarrow N}$, $\mathbf{U}_{n\leftarrow 2}$, and $\mathbf{W}_{n\leftarrow N}$ use $\ell_{2\leftarrow N}$, $\ell_{n\leftarrow 2}$, and $\ell_{n\leftarrow N}$ levels, respectively, with $\ell_{n\leftarrow 2} = 0$ when $n = 2$; EvalMod uses ℓ_{em} levels. Set $\ell = \ell_{N\leftarrow n}$ and $\tilde{\ell} = L - \ell_{2\leftarrow N} - \ell_{\mathsf{em}} - \ell_{n\leftarrow 2}$ and require $0 \leqslant \tilde{\ell} \leqslant L - \ell_{n\leftarrow N}$. We use key-switching keys from s to s' at level 0 and back at levels $\tilde{\ell} + \ell_{n\leftarrow N}$ and L. The overflow bound matches that of lazy flattening, while the remaining error comes from the new key switches, linear transformations, and EvalMod. Its size depends on EvalMod's input range and evaluation accuracy. See Supplementary Material A.6 for exact bounds and EvalMod input conditions.

7 Conversion Between Arithmetic and Digits

Throughout this section, we assume $n < N$, so that all n coefficients of a lane fit in the real slots of one ciphertext. For $t = X - \beta$ with an integer $\beta \geqslant 2$, the message space is $\mathcal{A}_n \cong \mathbb{Z}_M$, where $M = \beta^n + 1$. We convert between one arithmetic ciphertext holding N/n messages and two ciphertexts, each holding all n digits of $N/(2n)$ of these messages in its slots. We use n base-β digits per message: β^n is represented by $(-1, 0, \ldots, 0)$, and all other residues by digits in $[\![\beta]\!]$, saving the extra digit required for a canonical representation. We adapt Wei *et al.*'s digit extraction and compression [WMGZ26, Section 3.2], using modulo-$(2^n + 1)$ adder techniques [Zim99, VD12] for alignment and final addition. Section 8.4 reports corresponding experiments.

Algorithm 2: ArithBoot

Input: $\mathbf{ct}_{\mathsf{in}}$ representing $\mathbf{a} \in \mathcal{A}_n^{N/n}$ at level ℓ.
Output: $\mathbf{ct}_{\mathsf{out}}$ representing $\mathbf{a}$ at level $\tilde{\ell}$, with refreshed overflow and error.

1: $\mathbf{ct} \leftarrow \mathsf{LinTrans}(\mathsf{Ecd}_{\mathbb{C}}(\mathbf{V}_{N\leftarrow n}), \mathbf{ct}_{\mathsf{in}})$	$\mathcal{S}_N$	0
2: $\mathbf{ct} \leftarrow \mathsf{KeySwitch}_{s'\leftarrow s}(\mathbf{ct})$	$\mathcal{S}_N$	0
3: $\mathbf{ct} \leftarrow \mathsf{ModRaise}_{\tilde{\ell}+\ell_{n\leftarrow N}\leftarrow 0}(\mathbf{ct})$	$\mathcal{S}_N$	$\tilde{\ell}+\ell_{n\leftarrow N}$
4: $\mathbf{ct} \leftarrow \mathsf{Mul}\big(t(X^{N/n}), \mathbf{ct}\big)$	$\mathcal{S}_N$	$\tilde{\ell}+\ell_{n\leftarrow N}$
5: $\mathbf{ct}' \leftarrow [\mathbf{ct}]_{q_0}$	$\mathcal{S}_N$	0
6: $\mathbf{ct}' \leftarrow \mathsf{ModRaise}_{L\leftarrow 0}(\mathbf{ct}')$	$\mathcal{S}_N$	L
7: $\mathbf{ct}' \leftarrow \mathsf{KeySwitch}_{s\leftarrow s'}(\mathbf{ct}')$	$\mathcal{S}_N$	L
8: $\mathbf{ct}' \leftarrow \mathsf{LinTrans}(\mathsf{Ecd}_{\mathbb{C}}(\mathbf{W}_{2\leftarrow N}), \mathbf{ct}')$	$\mathcal{S}_2^{N/2}$	$L-\ell_{2\leftarrow N}$
9: $\mathbf{ct}' \leftarrow \mathsf{EvalMod}(\mathbf{ct}')$	$\mathcal{S}_2^{N/2}$	$L-\ell_{2\leftarrow N}-\ell_{\mathsf{em}}$
10: $\mathbf{ct}' \leftarrow \mathsf{LinTrans}(\mathsf{Ecd}_{\mathbb{C}}(\mathbf{U}_{n\leftarrow 2}), \mathbf{ct}')$	$\mathcal{S}_n^{N/n}$	$\tilde{\ell}$
11: $\mathbf{ct} \leftarrow \mathsf{KeySwitch}_{s\leftarrow s'}(\mathbf{ct})$	$\mathcal{S}_N$	$\tilde{\ell}+\ell_{n\leftarrow N}$
12: $\mathbf{ct} \leftarrow \mathsf{LinTrans}(\mathsf{Ecd}_{\mathbb{C}}(\mathbf{W}_{n\leftarrow N}), \mathbf{ct})$	$\mathcal{S}_n^{N/n}$	$\tilde{\ell}$
13: $\mathbf{ct}_{\mathsf{out}} \leftarrow \mathsf{Add}\big(\mathbf{ct}, \mathsf{Mul}(-1, \mathbf{ct}')\big)$	$\mathcal{S}_n^{N/n}$	$\tilde{\ell}$
14: **return** $\mathbf{ct}_{\mathsf{out}}$		

7.1 Arithmetic-to-Digit Conversion

Exposing the Coefficients. A lane contains $P = \sum_{k=0}^{n-1} P_k X^k = \mathsf{Lift}_n(a) + (X-\beta)I$, whose evaluation at $X = \beta$ represents $a \in \mathcal{A}_n$. Its coefficients may be negative or exceed $\beta - 1$. Partial $\mathsf{CoeffToSlot}$, implemented by $\mathbf{U}_{2\leftarrow n}$, exposes these coefficients in real and imaginary slot components. We then bit-reverse the $N/2$ slots, reversing the $\log(N/2)$ bits of each slot index. For each lane, the resulting slots hold $P_i + \mathrm{i}P_{i+n/2}$ for $i \in [\![n/2]\!]$, in increasing order and spaced N/n slots apart. Thus each lane's lower and upper coefficient halves still occupy real and imaginary components, respectively. We next rearrange the slots and exchange components between them so that all n coefficients of each lane occupy components of the same kind. With $\nu = N/(2n)$, the target layout places the coefficient P_i of lane j in the real component of slot $i\nu + j$, and that of lane $j + \nu$ in its imaginary component, for $i \in [\![n]\!]$ and $j \in [\![\nu]\!]$. Separating the two components gives two ciphertexts, each holding the complete coefficient vectors $\mathbf{z} = (P_i)_{i\in[\![n]\!]}$ of ν lanes.

These steps admit an efficient homomorphic implementation. We compose the bit reversal and the subsequent slot permutation, then factor the combined permutation into a single Beneš network, following Halevi and Shoup [HS14, Section 3]. Each layer has at most three nonzero cyclic diagonals; grouping consecutive layers trades the number of levels for the cost per group, as for partial $\mathsf{CoeffToSlot}$ and $\mathsf{SlotToCoeff}$. GZ and related constructions need no extra permutation: their custom linear maps from the quotient ring to slots directly yield natural order.

For an integer vector $\mathbf{u} = (u_i)_{i\in[\![n]\!]} \in \mathbb{Z}^n$, define its value in $\mathbb{Z}_M$ by:

$$\mathsf{val}_\beta(\mathbf{u}) = \Big(\sum_{i=0}^{n-1} u_i \beta^i\Big) \bmod M.$$

Thus $\mathsf{val}_\beta(\mathbf{z}) = a$. Subsequent operations act on all lanes of a ciphertext in parallel. We describe one lane; its entries are spaced ν slots apart and fill one cycle of the slots, so the rotation Rot_j of a lane, with indices modulo n, is a slot rotation by $j\nu$, and the entries that leave at one end re-enter at the other. Keeping a message within one ciphertext requires the rearrangement above, but makes the wraparound required modulo M a masked rotation.

Extracting and Aligning Digits. We first add a public offset to make the coefficients nonnegative. Given a public integer bound $\|P\|_\infty \leqslant B_P$, put $B_P^+ = 2B_P + 2\lceil \beta/2 \rceil$ and $W = \lceil \log_\beta(B_P^+ + 1) \rceil$, and assume $W \leqslant n$. Applying lazy flattening or arithmetic bootstrapping beforehand reduces B_P and hence W. Both procedures use $\mathsf{ModRaise}$ under an ephemeral ternary secret of Hamming weight h'. In the low-error regime, integrality gives $\|P\|_\infty \leqslant \lfloor (\beta+1)(h'+1)/2 \rfloor$ (see the proof of Proposition 10). For $h' = 32$ and $\beta = 2$, we therefore take $B_P = 49$, giving $B_P^+ = 100$ and $W = 7$. To construct the offset, start with a public polynomial whose coefficients all equal $B_P + \lceil \beta/2 \rceil$. Then add the lift of a public residue chosen so that the offset represents $-\gamma - 1$ modulo M, where γ is a public correction term that will be defined below. By (1) and integrality, this correction changes each coefficient by at most $\lfloor (\beta+1)/2 \rfloor = \lceil \beta/2 \rceil$. Adding the offset to P therefore puts its coefficients in $[\![0, B_P^+]\!]$ and changes its represented value from a to $a - \gamma - 1$. The alignment, compression, and final addition below add back $\gamma + 1$.

Following [WMGZ26, Section 3.2], we extract each shifted coefficient's W base-β digits using one multi-value functional bootstrapping call per ciphertext [DAP+26, Section 3.1]. This gives W vectors $\mathbf{z}_r$ for $r \in [\![W]\!]$, where vector r has weight β^r. To incorporate this weight explicitly, we shift vector r by r positions toward higher weights. Digits crossing the top must change sign because $\beta^n \equiv_M -1$; we replace each such digit x by $\beta - 1 - x$ to keep it in $[\![\beta]\!]$, and let it re-enter at the bottom. For an input vector $\mathbf{u} = (u_i)_{i \in [\![n]\!]}$, this defines the output $\mathbf{w} = (w_i)_{i \in [\![n]\!]} = \mathsf{align}_r(\mathbf{u})$, whose entries are $w_i = u_{i-r}$ for $i \geqslant r$ and $w_i = \beta - 1 - u_{n-r+i}$ for $i < r$. The output therefore satisfies:

$$\mathsf{val}_\beta(\mathbf{w}) = \beta^r \mathsf{val}_\beta(\mathbf{u}) + \beta^r - 1.$$

The added $\beta^r - 1$ is the price of using complements instead of negative digits. Homomorphically, apply Rot_{-r}, multiply entrywise by a public vector whose first r entries are -1 and whose remaining entries are 1, then add $\beta - 1$ to the first r entries. Applying align_r to $\mathbf{z}_r$ gives the aligned vector $\mathbf{w}_r$ for each $r \in [\![W]\!]$.

Algorithm 3: FinalDigitComb

Input: $\mathbf{u}_\ell \in [\![\beta]\!]^n$ for $\ell \in [\![2]\!]$ with $\mathsf{val}_\beta(\mathbf{u}_0) + \mathsf{val}_\beta(\mathbf{u}_1) = a - 1$.
Output: $\mathbf{d} \in \mathbb{Z}^n$ with entries in $[\![\beta]\!]$ and $\mathsf{val}_\beta(\mathbf{d}) = a$ when $a \neq \beta^n$; otherwise, $\mathbf{d} = -\mathbf{b}_0$.

$\mathbf{v} \leftarrow \mathbf{u}_0 + \mathbf{u}_1$
$\boldsymbol{\sigma} \leftarrow \mathsf{CarryType}(\mathbf{v})$
$\boldsymbol{\sigma}^{(0)} \leftarrow \boldsymbol{\sigma} + \mathbf{b}_0 \odot \big(\boldsymbol{\sigma} \odot (\boldsymbol{\sigma} - \mathbf{1}) - \boldsymbol{\sigma}\big), \quad \boldsymbol{\sigma}^{(1)} \leftarrow \boldsymbol{\sigma} + \mathbf{b}_0 \odot \big(\boldsymbol{\sigma} \odot (3 \cdot \mathbf{1} - \boldsymbol{\sigma}) - \boldsymbol{\sigma}\big)$
for $i \in [\![\log(n)]\!]$ **do**
    $r \leftarrow 2^i$
    for $\boldsymbol{\tau} \in \{\boldsymbol{\sigma}^{(0)}, \boldsymbol{\sigma}^{(1)}, \boldsymbol{\sigma}\}$ **do**
        $\boldsymbol{\tau} \leftarrow \mathsf{MergeCarry}(\mathsf{Rot}_{-r}(\boldsymbol{\tau}), \boldsymbol{\tau})$
    end for
end for
$\mathbf{c} \leftarrow \mathsf{EvalCarry}(\mathbf{0}, \boldsymbol{\sigma}^{(0)}) + \mathsf{EvalCarry}(\mathbf{0}, \boldsymbol{\sigma}^{(1)}) - \mathsf{EvalCarry}(\mathbf{0}, \boldsymbol{\sigma})$
$\mathbf{c}' \leftarrow \mathsf{align}_1(\mathbf{c})$
$\mathbf{d} \leftarrow \mathbf{v} - \beta\mathbf{c} + \mathbf{c}' - (\beta - 2)\mathbf{b}_0$
return $\mathbf{d}$

Combining W Vectors into Two. Starting from these W aligned vectors $\mathbf{w}_r$, we combine three current vectors at a time using a 3-to-2 compressor [WMGZ26, Section 3.2]. For each position, let x, y, z denote the digits at that position in the three input vectors. We write $x + y + z = s + \beta c$, where $c = \lfloor (x + y + z)/\beta \rfloor$ is the carry digit and $s = x + y + z - \beta c$ is the sum digit, both in $[\![\beta]\!]$.

Homomorphically, we compute c by evaluating at $x+y+z$ the polynomial interpolating $v \mapsto \lfloor v/\beta \rfloor$ on $[\![0, 3\beta-3]\!]$, whose degree is at most $3\beta-3$. For $\beta=2$, we can also use $c = yz + (y+z-2yz)x$. Alternatively, functional bootstrapping [AKP25] can evaluate the same look-up table (LUT) while refreshing the carry ciphertexts. We replace the three input vectors by the sum vector, whose digits stay in place, and the aligned carry vector obtained by applying align_1 to the carry vector. We repeat this procedure until only two vectors remain. Each compression therefore adds $\beta-1$ to the represented sum. Previously, constructing each aligned vector $\mathbf{w}_r$ from $\mathbf{z}_r$ added $\beta^r - 1$. Together with the $W-2$ compressions, the total increment is $\gamma = \sum_{r=0}^{W-1}(\beta^r-1) + (\beta-1)(W-2)$. Our initial offset cancels this increment. The two remaining vectors, $\mathbf{u}_\ell$ for $\ell \in [\![2]\!]$, satisfy:

$$\mathsf{val}_\beta(\mathbf{u}_0) + \mathsf{val}_\beta(\mathbf{u}_1) = a - 1.$$

The remaining offset of -1 is intentional: the final addition below corrects it.

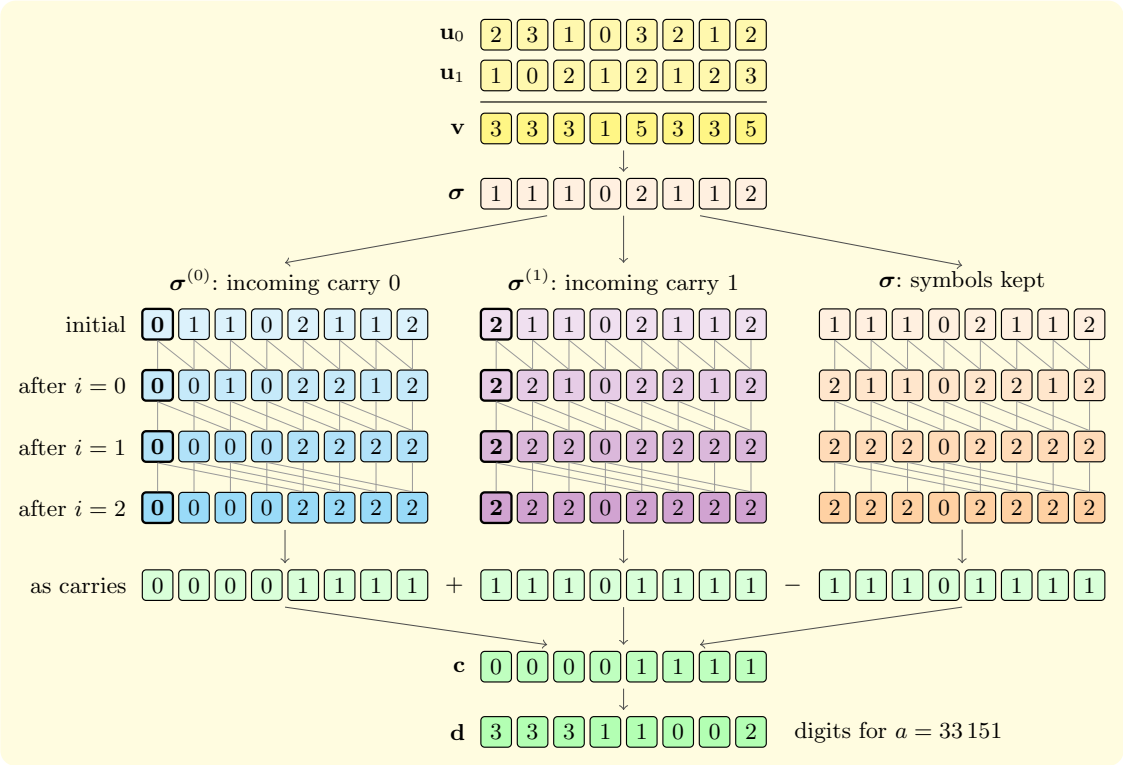


Fig. 5. FinalDigitComb (Algorithm 3) for $\beta=4$ and $n=8$, with entries ordered by increasing radix weight. From left to right, the branches process $\boldsymbol{\sigma}^{(0)}$, $\boldsymbol{\sigma}^{(1)}$, and $\boldsymbol{\sigma}$; bold initial cells mark the fixed symbols. Diagonal edges supply the lower block at distance $r=2^i$; wraparound edges are omitted. Applying $\mathsf{EvalCarry}(\mathbf{0}, \cdot)$ gives the three carry rows below the loop; adding the first two and subtracting the third yields $\mathbf{c}$.

Final Addition. FinalDigitComb (Algorithm 3) combines the remaining two vectors by adapting the parallel carry computation of LCtoC [CPL26, Section 3.2, Algorithm 4] and inverted end-around carry [Zim99, Section 2.4]. For a sum x of two digits and an incoming carry $c \in \{0,1\}$, the outgoing carry is $\lfloor (x+c)/\beta \rfloor$. We encode its behavior by the symbol $\sigma = \mathsf{CarryType}(x)$: $\sigma=0$ if $x < \beta-1$ (outgoing carry 0), $\sigma=1$ if $x=\beta-1$ (outgoing carry c), and $\sigma=2$ if $x \geqslant \beta$ (outgoing

carry 1). Homomorphically, we evaluate CarryType using one functional bootstrapping call per ciphertext [AKPZ26, CPL26]; for $\beta = 2$, it is the identity, and this call can be omitted if no refreshing is needed. Given an incoming carry c, EvalCarry converts symbols $\sigma = 0, 1, 2$ into outgoing carries $0, c, 1$, respectively:

$$\mathsf{EvalCarry}(c, \sigma) = \frac{\sigma(\sigma - 1)}{2} + c\,\sigma(2 - \sigma).$$

The same symbols $\sigma = 0, 1, 2$ apply to a block of consecutive digits: after carry propagation through the entire block, the outgoing carry is always 0, equals the incoming carry c, or is always 1, respectively. For two adjacent blocks, let σ_{lo} and σ_{hi} be these symbols for the lower and higher block, respectively. MergeCarry combines the symbols σ_{lo} and σ_{hi} in this order: if $\sigma_{\mathsf{hi}} = 1$, the higher block passes its incoming carry through, so keep σ_{lo}; otherwise, keep σ_{hi}. The corresponding polynomial is:

$$\mathsf{MergeCarry}(\sigma_{\mathsf{lo}}, \sigma_{\mathsf{hi}}) = \sigma_{\mathsf{hi}}(\sigma_{\mathsf{hi}} - 1) + \sigma_{\mathsf{lo}}\sigma_{\mathsf{hi}}(2 - \sigma_{\mathsf{hi}}).$$

All maps act entrywise on vectors of length n. Put $\mathbf{b}_0 = (1, 0, \ldots, 0)$.

Algorithm 3 combines the two vectors representing $a - 1$ into the digits of a. To account for both the modulus $M = \beta^n + 1$ and the missing $+1$, the initial digit must receive the complement of the carry leaving the last digit. The loop computes three auxiliary carry vectors in parallel; adding those obtained from $\boldsymbol{\sigma}^{(0)}$ and $\boldsymbol{\sigma}^{(1)}$ and subtracting the one obtained from $\boldsymbol{\sigma}$ gives the required complemented wraparound. The final alignment and correction then yield the output digits. Figure 5 illustrates the full computation for $\beta = 4$ and $n = 8$, from the input vectors through the three carry computations to the output digits. We provide further details in Supplementary Material A.7.

7.2 Digit-to-Arithmetic Conversion

The two input ciphertexts together hold the digit vectors of all N/n lanes. We combine their slots as real and imaginary components of one ciphertext, undo the component exchange and the slot permutations, and apply partial SlotToCoeff via $\mathbf{U}_{n\leftarrow 2}$. This places each lane's digits into the coefficients of its polynomial, which represents the same message modulo $X - \beta$. The result is one arithmetic ciphertext containing all lanes. For the message β^n, the exceptional digit vector $(-1, 0, \ldots, 0)$ becomes the constant polynomial -1, which represents the same residue modulo $M = \beta^n + 1$.

7.3 Homomorphic Cost

For fixed β and ephemeral secret weight h', preliminary lazy flattening or arithmetic bootstrapping makes B_P and hence W independent of n. We exclude this refreshing and count bootstrapping separately. Grouping partial transforms into at most $\rho \geqslant 1$ balanced groups costs $\mathcal{O}(\rho n^{1/\rho})$ ring operations and $\mathcal{O}(\rho)$ levels. The two halves of the permutation's Beneš network have monotone power-of-two strides; grouping each half similarly costs $\mathcal{O}(\rho N^{1/\rho})$ operations and $\mathcal{O}(\rho)$ levels [HS14, Sections 3.2 and 3.4]. The carry loop adds $\mathcal{O}(\log n)$ operations and levels; all remaining steps outside bootstrapping cost $\mathcal{O}(1)$. Thus arithmetic-to-digit conversion costs $\mathcal{O}(\rho N^{1/\rho} + \log n)$ operations and $\mathcal{O}(\rho + \log n)$ levels per input ciphertext, while the reverse conversion costs $\mathcal{O}(\rho N^{1/\rho})$ operations and $\mathcal{O}(\rho)$ levels. With the same transform grouping, Wei *et al.*'s arithmetic-to-Boolean conversion [WMGZ26] costs $\mathcal{O}(\rho N^{1/\rho} + n)$ operations beyond bootstrapping, while their reverse conversion costs $\mathcal{O}(n)$. The linear term comes from their dense maps.

Digit extraction uses two multi-value functional bootstrapping calls, each returning W ciphertexts. CarryType uses two further calls [AKPZ26, CPL26], optional for $\beta = 2$ if no refreshing is needed; our implementation retains them to restore levels, giving four calls per input ciphertext (Section 8.4). Optional bootstrapping for the compressors or EvalCarry adds only $\mathcal{O}(1)$ calls, so the total remains $\mathcal{O}(1)$; the reverse conversion uses none.

8 Experimental Results

We evaluate our Lattigo implementation on AWS using the setup of Section 8.1. Section 8.2 compares refreshing time, memory, and operation costs with our GZ port across lane degrees, and tests further integer and finite-field message spaces. Section 8.3 measures error growth during repeated multiplication and estimates amortized costs including refreshing. Section 8.4 compares arithmetic-to-digit conversion costs per call and per message with our implementation of Wei *et al.*'s method [WMGZ26].

8.1 Setup

Platform and Methodology. All benchmarks in this section use RLWE dimension $N = 2^{16}$ and target 128-bit security. We run them on AWS Graviton5 `r9g.4xlarge` and `r9g.8xlarge` instances with 128 GiB and 256 GiB of memory, respectively. Our implementation and the GZ port use Go 1.24 and Lattigo v6.2.0 [Lat24]. Each benchmark uses one thread; times are means of ten iterations after warm-up unless stated otherwise, and memory measurements give the heap in use after the timed iterations. For a ciphertext with actual scale Δ', we report the base-two logarithm $\log(\|\mathbf{e}\|_\infty/\Delta')$ of its normalized coefficient error, taking the maximum over all coefficients and lanes. Refreshing restores the fixed scale $\Delta' = \Delta$.

Our Implementation. We implement lifting and packing, lazy flattening, and arithmetic bootstrapping in Lattigo for $t = X^r - c$, giving $\mathcal{A}_n \cong \mathbb{Z}_M[X]/\langle X^r - c\rangle$ with $M = c^{n/r} + 1$. This covers modular integer rings, extension fields, and their products; the non-binomial prime fields of Table 2 are not benchmarked. Section 8.4 describes digit conversion. After each multiplication and arithmetic bootstrapping, decryption is checked against an independent product. We evaluate the factorizations of Section 3 with baby-step giant-step and hoisted rotations [HS18], computing diagonals at 256-bit precision. A sweep selects the fastest admissible level split for each n (Table 8 in the supplementary material).

Parameters and Security. For refreshing and sustained arithmetic, we use scale $\Delta = 2^{60}$ for a large error budget between refreshings, with a chain of 56- and 60-bit primes and three 61-bit auxiliary primes. EvalMod uses range $K = 16$, a degree-30 approximation, and three double-angle steps. The finite-field benchmarks change t while retaining the scale, modulus chain, and transformation depths of their paired integer benchmarks (Section 8.2). Arithmetic-to-digit conversion uses a scale of 46 to 50 bits, a separate modulus chain, and four 55-bit auxiliary primes for functional bootstrapping (Section 8.4). Let $P_{\mathsf{aux}} = \prod_i p_{\mathsf{aux},i}$ denote the product of the auxiliary key-switching primes. For arithmetic refreshing, our main secret s is uniform ternary, with $\log(q_L P_{\mathsf{aux}})$ ranging from 1 119 to 1 175, below the 1 747-bit security limit [BCC+25]. The ephemeral secret s' is sparse ternary of Hamming weight $h' = 32$ and appears only modulo $q_0 p_{\mathsf{aux},0}$, with $\log(q_0 p_{\mathsf{aux},0}) = 121$; the estimator

of [HJO26] gives 161.46 bits of security. Supplementary Material B.1 analyzes overflow for these parameters.

8.2 Refreshing Time and Memory

Comparison with GZ. We port GZ's C++ prototype from OpenFHE [ABAB+22] to Lattigo and run both schemes on the same AWS hardware. We use $t = X - 2$, so both pack N/n messages per ciphertext, modulo $2^n + 1$ (ours) or 2^n (GZ). The port retains triangle encoding, the blockwise maps between slots and polynomial representatives, diagonal evaluation with hoisted rotations, three CoeffToSlot and two SlotToCoeff levels, and sparse-secret encapsulation. Its A2A applies overflow refreshing A2A-I followed by error refreshing A2A-e. We retain GZ's published parameters [GZ26]: $\Delta = 2^{43}$, 21 ciphertext primes of 43 bits, and seven auxiliary primes of 50 bits. Its degree-32 EvalMod approximation uses the same range and number of double-angle steps as ours. The sparse ternary main and ephemeral secrets have weights $h = 192$ and $h' = 32$, with $\log(q_L P_{\mathsf{aux}}) = 1\,253$ and $\log(q_0 p_{\mathsf{aux},0}) = 93$, respectively.

For $8 \leqslant n \leqslant 1\,024$, arithmetic bootstrapping is 2.1× to 6.8× faster than A2A, and lazy flattening is 3.8× to 65× faster than A2A-I (Table 4, Figure 2). Over $8 \leqslant n \leqslant 2^{14}$, our arithmetic bootstrapping stays nearly constant at 6.20 to 7.85 seconds, while lazy flattening falls from 1.78 to 0.13 seconds. Our memory use stays between 5 and 12 GiB, following the numbers of Galois keys and encoded transformations. The port's grows from 17 GiB at $n = 8$ to 157 GiB at $n = 1\,024$, preventing measurements at larger n: its blockwise maps store a number of encoded diagonals proportional to n. For $8 \leqslant n \leqslant 256$, the port's A2A-$I$ and A2A are 1.3 to 1.7 times slower than the published OpenFHE timings on x86 with Intel HEXL [GZ26] (Supplementary Material B.5).

Table 4. Single-threaded Lattigo refreshing times on AWS at $N = 2^{16}$: our LazyFlatten and ArithBoot ($t = X - 2$) versus the GZ port's A2A-I and A2A. The A2A-I time at $\log n = 10$ is a single run; the port does not fit in the 256 GiB memory of our instance beyond this degree.

$\log n$	N/n	Overflow refresh			Overflow & error refresh		
		LazyFlatten	A2A-I	Gain	ArithBoot	A2A	Gain
3	8 192	1.78 s	6.78 s	3.8×	7.85 s	16.80 s	2.1×
4	4 096	1.43 s	6.95 s	4.9×	7.54 s	17.03 s	2.3×
5	2 048	1.23 s	8.11 s	6.6×	7.47 s	18.85 s	2.5×
6	1 024	0.82 s	8.62 s	10.5×	7.04 s	19.71 s	2.8×
7	512	0.74 s	11.25 s	15.1×	6.98 s	23.96 s	3.4×
8	256	0.64 s	12.88 s	20.1×	7.00 s	26.76 s	3.8×
9	128	0.56 s	19.43 s	35.0×	7.04 s	37.38 s	5.3×
10	64	0.39 s	25.35 s	65.0×	6.91 s	47.11 s	6.8×
11	32	0.28 s	–	–	6.20 s	–	–
12	16	0.22 s	–	–	6.44 s	–	–
14	4	0.13 s	–	–	6.76 s	–	–

In Figure 6, $\mathbf{W}_{2 \leftarrow N}$ and EvalMod account for 73% to 82% of our time, with sizes and parameters independent of n (step timings: Table 12 in the supplementary material). Fully factored, $\mathbf{V}_{N \leftarrow n}$ and $\mathbf{W}_{n \leftarrow N}$ have $\log(N/n)$ elementary factors each, and $\mathbf{U}_{n \leftarrow 2}$ has $\log(n/2)$. As n grows, the first two shorten and the third lengthens, so their costs essentially cancel. Lazy flattening uses only the first two, hence its decreasing latency. GZ's extra slot–polynomial maps cost 2.0 seconds at $n = 8$ and 22.5 seconds at $n = 512$, driving the growing gap.

Other Message Spaces. Table 5 pairs finite-field moduli $t = X^r - c$ with integer baselines $t = X - c$ at the same n, c, scale, chain, and transformation depths. Their refreshing times closely match: changing t affects only $\mathbf{D}_t$ and $\mathbf{D}_t^{-1}$, not the elementary transformation factors. Against [SS26] on the same AWS instance with its own parameters, ours is faster per element for $\mathbb{F}_{5^{16}}$ and $\mathbb{F}_{37^{32}}$, but slower for $\mathbb{F}_{17^4}$, where their implementation refreshes four times as many elements per call. Their parameters do not support $p = 257$. An additional experiment varies $t = X - \beta$ over $\beta \in \{2, 8, 26, 64\}$ at $n = 8$ and $n = 4\,096$, with fixed EvalMod parameters. Arithmetic bootstrapping times vary by about 2%, with logarithmic normalized errors between -35.5 and -34.8 (Table 10 in the supplementary material); at $n = 4\,096$ and $\beta = 64$, it refreshes 16 integers modulo $2^{24\,576} + 1$ in 6.34 seconds.

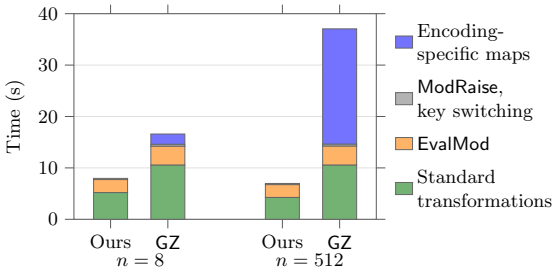


Fig. 6. Cost of arithmetic bootstrapping on AWS at $N = 2^{16}$, at the same lane degree for both schemes (N/n messages modulo $2^n + 1$ for ours and modulo 2^n for GZ). Profiling totals differ from Table 4 by about 1%.

Table 5. Refreshing times on AWS at $N = 2^{16}$, averaged over three iterations, for extension fields and the corresponding integer message spaces. Each of our lanes holds one element for $\mathbb{F}_{5^{16}}$ and $\mathbb{F}_{37^{32}}$, two for $\mathbb{F}_{17^4}$, and sixteen for $\mathbb{F}_{257^8}$. The last columns compare multiplication followed by bootstrapping per field element with [SS26] at its own parameters.

Field	$\log n$	Finite-field messages			Modular-integer baseline			Per element	
		t	LazyFlatten	ArithBoot	t	LazyFlatten	ArithBoot	Ours	[SS26]
$\mathbb{F}_{5^{16}}$	5	$X^{16} - 2$	1.21 s	7.50 s	$X - 2$	1.21 s	7.47 s	3.68 ms	4.99 ms
$\mathbb{F}_{17^4}$	5	$X^8 - 2$	1.21 s	7.52 s	$X - 2$	1.21 s	7.47 s	1.84 ms	1.21 ms
$\mathbb{F}_{37^{32}}$	6	$X^{32} - 6$	0.81 s	7.09 s	$X - 6$	0.81 s	6.96 s	6.96 ms	12.76 ms
$\mathbb{F}_{257^8}$	10	$X^{128} - 2$	0.39 s	6.96 s	$X - 2$	0.39 s	6.87 s	6.83 ms	–

8.3 Sustained Computation: Error and Amortized Cost

Evaluation Schedule. We study repeated squaring with the integer parameters of Section 8.1 and the GZ baseline of Section 8.2, starting from a refreshed ciphertext. After each multiplication, lazy flattening or A2A-I controls overflow and restores levels; every k multiplications, arithmetic bootstrapping or A2A-e refreshes the error, respectively. We choose the largest k for which all 50 tested cycles decrypt correctly and return to the observed error floor. Writing T_{mul}, T_{lazy}, T_{boot}, $T_{\text{A2A-}I}$, and $T_{\text{A2A-}e}$ for the measured operation times, the estimated amortized costs per multiplication are $T_{\mathsf{mul}} + T_{\mathsf{lazy}} + T_{\mathsf{boot}}/k$ and $T_{\mathsf{mul}} + T_{\text{A2A-}I} + T_{\text{A2A-}e}/k$ for our scheme and GZ, respectively.

Results. At $n = 8$, the logarithmic normalized errors start near -35.4 (ours) and -23.4 (GZ) and grow by about 5 and 6 bits per multiplication. This permits four and two multiplications per cycle, respectively; one more prevents error refreshing from restoring the floor. Figure 7 plots this behavior using measured operation times. The cycle lengths and estimated costs in Table 6 show nearly constant amortized latency per ciphertext for ours across the tested degrees, while the port's grows.

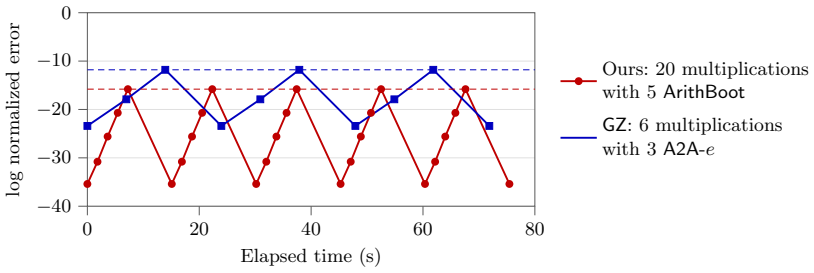


Fig. 7. Log normalized coefficient error during repeated squaring at $n = 8$, with elapsed time reconstructed from Table 4. Overflow is refreshed after each multiplication and error at each cycle's end. Dashed lines mark the largest tested errors from which error refreshing restored the floor.

8.4 Arithmetic-to-Digit Conversion

Implementation and Parameters. We implement Section 7's arithmetic-to-digit conversion for $\beta = 2$, but not its inverse. Lazy flattening gives inputs with $B_P = 49$ and $W = 7$. Each of the two ciphertexts then uses one multi-value functional bootstrapping call to extract W bits per exposed coefficient and one functional bootstrapping call for CarryType. We call the ciphertext-modulus chain used for homomorphic arithmetic and refreshing the *arithmetic chain*. Digit conversion uses a separate *conversion chain*, which shares its lowest primes and the main secret s with it. A ciphertext at a shared level can therefore be used unchanged with either parameter set. The conversion chain adds smaller primes, of 46 to 50 bits, to provide the required levels (Supplementary Material B.1); its modulus size $\log(q_L P_{\mathsf{aux}})$ stays below the 1 747-bit limit for 128-bit security [BCC+25].

Baseline and Measurement. Our Lattigo baseline uses Wei *et al.*'s conversion [WMGZ26], parameters, and modulus 2^n. Timings start with overflow and include lazy flattening, coefficient exposure, and four functional bootstrapping calls for ours; A2A-I and two calls for the baseline. The baseline uses only real slot components and n zero slots per n-digit message to isolate carries, converting $N/(4n)$ messages per call, one quarter of the input. Comparisons concern these implementations; the baseline omits some optimizations.

Results. For $64 \leqslant n \leqslant 1\,024$, our time per call varies by about 11% and is comparable to the baseline's (1.2× to 1.6× slower); per converted message, ours is 2.5× to 3.2× faster (Table 7). Up to $n = 2^{15}$, our time per call grows by at most about 4% for each doubling of n. All recovered digits match the independently computed products modulo $2^n + 1$.

Table 6. Cycle lengths k and estimated amortized multiplication times on AWS at $N = 2^{16}$ and $t = X - 2$, using the formulas above. Each multiplication is followed by lazy flattening or A2A-I; each cycle ends with arithmetic bootstrapping or A2A-e. The GZ port does not fit in the 256 GiB memory of our instance at $\log n = 12$.

$\log n$	Ours			GZ			Gain
	k	Per ciphertext	Per message	k	Per ciphertext	Per message	
3	4	3.77 s	0.46 ms	2	11.98 s	1.46 ms	3.2×
6	3	3.20 s	3.13 ms	1	19.86 s	19.4 ms	6.2×
9	3	2.94 s	23.0 ms	1	37.55 s	293 ms	12.8×
12	2	3.47 s	217 ms	–	–	–	–

Table 7. Arithmetic-to-digit conversion from a ciphertext with overflow at $N = 2^{16}$. Ours uses modulus $2^n + 1$; the baseline is our implementation of [WMGZ26] modulo 2^n. Timings are means of ten single-threaded runs on AWS. The baseline is not run beyond $n = 2^{10}$, where its peak memory already reaches 232 GiB.

$\log n$	Ours			[WMGZ26]		
	Converted messages	Per call	Per message	Converted messages	Per call	Per message
6	1 024	113.3 s	110.7 ms	256	70.1 s	273.9 ms
7	512	114.6 s	223.8 ms	128	75.1 s	586.6 ms
8	256	117.7 s	459.8 ms	64	74.9 s	1 170 ms
9	128	122.1 s	953.9 ms	32	91.1 s	2 846 ms
10	64	126.2 s	1 971 ms	16	102.3 s	6 395 ms
11	32	124.7 s	3 898 ms	–	–	–
12	16	129.0 s	8 065 ms	–	–	–
13	8	134.0 s	16 751 ms	–	–	–
14	4	139.6 s	34 898 ms	–	–	–
15	2	144.8 s	72 420 ms	–	–	–

Supplementary Material

Supplementary Material A provides correctness proofs and error bounds. Supplementary Material B gives implementation parameters and details, operation counts, and additional experiments on precision, conversion correctness, and MacBook Pro performance. Under the IACR policy, this work falls into the "Limited or no AI contribution" category. We used large language models for writing, consistency checks, and implementation support; Supplementary Material C discloses the models, uses, and representative requests.

Acknowledgments. The last author acknowledges the support of the Luxembourgish "Fonds National de la Recherche" (FNR) through an Individual Grant (reference number: 17936291).

Disclosure of Interests. The authors declare no competing interests.

References

ABAB+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, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

AKP25. Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General Functional Bootstrapping Using CKKS. In *Advances in Cryptology – CRYPTO 2025*, pages 304–337. Springer, 2025. doi:10.1007/978-3-032-01881-6_10.

AKPZ26. Andreea Alexandru, Andrey Kim, Yuriy Polyakov, and Hongren Zheng. Sparse Hermite Interpolation Method for Discrete-CKKS Functional Bootstrapping. Cryptology ePrint Archive, Paper 2026/1026, 2026. https://eprint.iacr.org/2026/1026.

BCC+25. 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. *IACR Communications in Cryptology*, 1(4), 2025. doi:10.62056/anxra69p1.

BCKS24. Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, and Damien Stehlé. Bootstrapping Bits with CKKS. In *Advances in Cryptology – EUROCRYPT 2024*, pages 94–123. Springer, 2024. doi:10.1007/978-3-031-58723-8_4.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012. doi:10.1145/2090236.2090262.

BK25. Dan Boneh and Jaehyung Kim. Homomorphic Encryption for Large Integers from Nested Residue Number Systems. In *Advances in Cryptology – CRYPTO 2025*, pages 338–370. Springer, 2025. doi:10.1007/978-3-032-01881-6_11.

BKSS25. Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping Small Integers with CKKS. In *Advances in Cryptology – ASIACRYPT 2024*, pages 330–360. Springer, 2025. doi:10.1007/978-981-96-0875-1_11.

Bra12. Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, pages 868–886. Springer, 2012. doi:10.1007/978-3-642-32009-5_50.

BTPH22. Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation. In *Applied Cryptography and Network Security*, pages 521–541. Springer, 2022. doi:10.1007/978-3-031-09234-3_26.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33. Springer, 2016. doi:10.1007/978-3-662-53887-6_1.

CHK+18. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT 2018*, pages 360–384. Springer, 2018. doi:10.1007/978-3-319-78381-9_14.

CHK+19. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography – SAC 2018*, pages 347–368. Springer, 2019. doi:10.1007/978-3-030-10970-7_16.

CHKS25. Jung Hee Cheon, Guillaume Hanrot, Jongmin Kim, and Damien Stehlé. SHIP: A Shallow and Highly Parallelizable CKKS Bootstrapping Algorithm. In *Advances in Cryptology – EUROCRYPT 2025*, pages 398–428. Springer, 2025. doi:10.1007/978-3-031-91131-6_14.

CK25. Jean-Sébastien Coron and Robin Köstler. Low-Latency Bootstrapping for CKKS using Roots of Unity. Cryptology ePrint Archive, Paper 2025/651, 2025. https://eprint.iacr.org/2025/651.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437. Springer, 2017. doi:10.1007/978-3-319-70694-8_15.

CLPX18. Hao Chen, Kim Laine, Rachel Player, and Yuhou Xia. High-Precision Arithmetic in Homomorphic Encryption. In *Topics in Cryptology – CT-RSA 2018*, pages 116–136. Springer, 2018. doi:10.1007/978-3-319-76953-0_7.

CPL26. Gyeongwon Cha, Dongjin Park, and Joon-Woo Lee. Improved Radix-Based Approximate Homomorphic Encryption for Large Integers via Lightweight Bootstrapped Digit Carry. In *Advances in Cryptology – EUROCRYPT 2026*, pages 243–273. Springer, 2026. https://eprint.iacr.org/2025/1740. doi:10.1007/978-3-032-25327-9_9.

CS26. Jean-Sébastien Coron and Tim Seuré. PaCo: Bootstrapping for CKKS via Partial CoeffToSlot. In *Advances in Cryptology – ASIACRYPT 2025*, pages 36–67. Springer, 2026. doi:10.1007/978-981-95-5122-4_2.

DAP+26. Jules Dumezy, Andreea Alexandru, Yuriy Polyakov, Pierre-Emmanuel Clet, Olive Chakraborty, and Aymen Boudguiga. Evaluating Larger Lookup Tables using CKKS. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(1):559–591, 2026. doi:10.46586/tches.v2026.i1.559-591.

DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640. Springer, 2015. doi:10.1007/978-3-662-46800-5_24.

DMPS24. Nir Drucker, Guy Moshkowich, Tomer Pelleg, and Hayim Shaul. BLEACH: Cleaning Errors in Discrete Computations Over CKKS. *Journal of Cryptology*, 37(1), 2024. Article 3. doi:10.1007/s00145-023-09483-1.

Für09. Martin Fürer. Faster Integer Multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. doi:10.1137/070711761.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

Gen09. Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM, 2009. doi:10.1145/1536414.1536440.

GV25. Robin Geelen and Frederik Vercauteren. Fully Homomorphic Encryption for Cyclotomic Prime Moduli. In *Advances in Cryptology – EUROCRYPT 2025*, pages 366–397. Springer, 2025. doi:10.1007/978-3-031-91131-6_13.

GZ26. Mingyu Gao and Hongren Zheng. FHE for SIMD Arithmetic Logic Units with Amortized $O(1)$ Bootstrapping per Ciphertext. In *Advances in Cryptology – CRYPTO 2026*, pages 299–330. Springer, 2026. doi:10.1007/978-3-032-35374-0_10.

HHC19. Kyoohyung Han, Minki Hhan, and Jung Hee Cheon. Improved Homomorphic Discrete Fourier Transforms and FHE Bootstrapping. *IEEE Access*, 7:57361–57370, 2019. doi:10.1109/ACCESS.2019.2913850.

HJO26. Jianhua Hou, Haodong Jiang, and Tabitha Ogilvie. Careful with the Ring! Concrete Hardness Gaps Between LWE and MLWE. In *Advances in Cryptology – CRYPTO 2026*, pages 462–497. Springer, 2026. doi:10.1007/978-3-032-35377-1_15.

HS14. Shai Halevi and Victor Shoup. Algorithms in HElib. In *Advances in Cryptology – CRYPTO 2014*, pages 554–571. Springer, 2014. doi:10.1007/978-3-662-44371-2_31.

HS18. Shai Halevi and Victor Shoup. Faster Homomorphic Linear Transformations in HElib. In *Advances in Cryptology – CRYPTO 2018*, pages 93–120. Springer, 2018. doi:10.1007/978-3-319-96884-1_4.

JL26. Daehyun Jang and Junho Lee. Arithmetic for Large-Characteristic Finite Fields in CKKS. Cryptology ePrint Archive, Paper 2026/1958, 2026. https://eprint.iacr.org/2026/1958.

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 *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, pages 4361–4375. ACM, 2024. doi:10.1145/3658644.3690375.

Kim25a. Jaehyung Kim. Bootstrapping GBFV with CKKS. Cryptology ePrint Archive, Paper 2025/888, 2025. https://eprint.iacr.org/2025/888.

Kim25b. Jaehyung Kim. Efficient Homomorphic Integer Computer from CKKS. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):873–898, 2025. doi:10.46586/tches.v2025.i4.873-898.

Kim26. Jaehyung Kim. Faster Homomorphic Integer Computer. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(3):1128–1151, 2026. doi:10.46586/tches.v2026.i3.1128-1151.

KLN+26. Taeseong Kim, Jonghoo Lee, Taeyeong Noh, Jung Hee Cheon, and Guillaume Hanrot. Latency-Aware, High-Throughput Homomorphic AES Evaluation with CKKS. Cryptology ePrint Archive, Paper 2026/1209, 2026. https://eprint.iacr.org/2026/1209.

KN25. Jaehyung Kim and Taeyeong Noh. Modular Reduction in CKKS. *IACR Communications in Cryptology*, 2(2), 2025. doi:10.62056/aevur-iuc.

KPK+22. Seonghak Kim, Minji Park, Jaehyung Kim, Taekyung Kim, and Chohong Min. EvalRound Algorithm in CKKS Bootstrapping. In *Advances in Cryptology – ASIACRYPT 2022*, pages 161–187. Springer, 2022. doi:10.1007/978-3-031-22966-4_6.

Lat24. Lattigo v6. Online: https://github.com/tuneinsight/lattigo, August 2024. EPFL-LDS, Tune Insight SA.

LM21. Baiyu Li and Daniele Micciancio. On the Security of Homomorphic Encryption on Approximate Numbers. In *Advances in Cryptology – EUROCRYPT 2021*, pages 648–677. Springer, 2021. doi:10.1007/978-3-030-77870-5_23.

LMSWS22. Baiyu Li, Daniele Micciancio, Mark Schultz-Wu, and Jessica Sorrell. Securing Approximate Homomorphic Encryption Using Differential Privacy. In *Advances in Cryptology – CRYPTO 2022*, pages 560–589. Springer, 2022. doi:10.1007/978-3-031-15802-5_20.

LMWW26. Jiaming Liu, Shihe Ma, Anyu Wang, and Xiaoyun Wang. High-Precision Homomorphic ALU over Arbitrary Moduli with $O(1)$ Bootstrapping. Cryptology ePrint Archive, Paper 2026/1869, 2026. https://eprint.iacr.org/2026/1869.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23. Springer, 2010. doi:10.1007/978-3-642-13190-5_1.

PZZ26. Chris Peikert, Doron Zarchy, and Guy Zyskind. High-Precision Exact FHE Made Simple, General, and Fast. In *Advances in Cryptology – CRYPTO 2026*, pages 525–557. Springer, 2026. doi:10.1007/978-3-032-35374-0_17.

SS26. Tim Seuré and Elias Suvanto. Finite-Field Arithmetic in CKKS. Cryptology ePrint Archive, Paper 2026/1102, 2026. https://eprint.iacr.org/2026/1102.

SSTX09. Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient Public Key Encryption Based on Ideal Lattices. In *Advances in Cryptology – ASIACRYPT 2009*, pages 617–635. Springer, 2009. doi:10.1007/978-3-642-10366-7_36.

VD12. Haridimos T. Vergos and Giorgos Dimitrakopoulos. On Modulo $2^n + 1$ Adder Design. *IEEE Transactions on Computers*, 61(2):173–186, 2012. doi:10.1109/TC.2010.261.

WMGZ26. Yuchen Wei, Kaisheng Ma, Mingyu Gao, and Hongren Zheng. FHE for ALU over Large Prime Moduli and Application to One-Round Threshold ECDSA. Cryptology ePrint Archive, Paper 2026/1836, 2026. https://eprint.iacr.org/2026/1836.

Zim99. Reto Zimmermann. Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication. In *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pages 158–167. IEEE, 1999. doi:10.1109/ARITH.1999.762841.

Supplementary Material

A Correctness and Error Bounds

We give worst-case overflow and error bounds and correctness proofs for the encryption, homomorphic operations, and refreshing procedures of Section 5 and Section 6. The last subsection explains the correctness of the final addition of the arithmetic-to-digit conversion (Section 7). We use the bounds $\|tF\|_\infty \leqslant \|t\|_1\|F\|_\infty$ and $\|FG\|_\infty \leqslant n\|F\|_\infty\|G\|_\infty$ for $F, G \in \mathcal{S}_n$, also applied componentwise to polynomial lanes.

A.1 Bounds on the Packing Isomorphisms

We use the 2-norm $\|\mathbf{z}\|_2 = \left(\sum_{i\in[\![n]\!]} |z_i|^2\right)^{1/2}$ for $\mathbf{z} \in \mathbb{C}^n$, extended to polynomials and tuples as in Section 2; for example, $\left\|(2+5X, 4-2\mathrm{i}X^2)\right\|_2 = \|(2,5,0,4,0,-2\mathrm{i})\|_2 = 7$.

Lemma 1 (Packing isomorphisms are scaled isometries). *For all divisors $d, d' \geqslant 2$ of N, the map $\sqrt{d'/d}\cdot\Phi_{d'\leftarrow d} : \mathcal{S}_d^{N/d} \to \mathcal{S}_{d'}^{N/d'}$ is an isometry with respect to the 2-norms on $\mathcal{S}_d^{N/d}$ and $\mathcal{S}_{d'}^{N/d'}$. In other words, we have $\|\Phi_{d'\leftarrow d}(\mathbf{P})\|_2 = \sqrt{d/d'}\cdot\|\mathbf{P}\|_2$ for every $\mathbf{P} \in \mathcal{S}_d^{N/d}$.*

Proof. The map $\sqrt{2/d}\cdot\mathsf{eval}_d : \mathcal{S}_d \to \mathbb{C}^{d/2}$ is an isometry with respect to the 2-norms on $\mathcal{S}_d$ and $\mathbb{C}^{d/2}$ by orthogonality of the roots of unity; see [CKKS17, Sections 2.2 and 3.2] for the canonical embedding. Consequently, the same holds true for the extended map $\sqrt{2/d}\cdot\mathsf{eval}_d : \mathcal{S}_d^{N/d} \to \mathbb{C}^{N/2}$ and the inverse map $\sqrt{d'/2}\cdot\mathsf{eval}_{d'}^{-1} : \mathbb{C}^{N/2} \to \mathcal{S}_{d'}^{N/d'}$, and hence for their composition $\sqrt{d'/d}\cdot\Phi_{d'\leftarrow d} : \mathcal{S}_d^{N/d} \to \mathcal{S}_{d'}^{N/d'}$. □

Lemma 2 (Infinity norms under packing isomorphisms). *For all divisors $d, d' \geqslant 2$ of N and every $\mathbf{P} \in \mathcal{S}_d^{N/d}$, we have $\|\Phi_{d'\leftarrow d}(\mathbf{P})\|_\infty \leqslant \sqrt{Nd/d'}\,\|\mathbf{P}\|_\infty$.*

Proof. Lemma 1 gives $\|\Phi_{d'\leftarrow d}(\mathbf{P})\|_\infty \leqslant \|\Phi_{d'\leftarrow d}(\mathbf{P})\|_2 = \sqrt{d/d'}\,\|\mathbf{P}\|_2 \leqslant \sqrt{Nd/d'}\,\|\mathbf{P}\|_\infty$. □

Remark 2. The bounds below hold uniformly for every power of two $n \in [\![2, N]\!]$. Sharper bounds are available for $n = N$, but we omit these refinements for readability.

A.2 Correct Decryption

We use the ciphertext representation and decoding map from Section 5. To simplify the correctness proofs below, we introduce the following notation. Given $\mathbf{a} \in \mathcal{A}_n^{N/n}$ and two bounds B_I and B_e, we let $\mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$ be the set of all ciphertexts $\mathbf{ct} \in \mathcal{R}_{N,q_\ell}^2$ for which there exist $\mathbf{I} \in \mathcal{R}_n^{N/n}$ and $\mathbf{e} \in \mathcal{S}_n^{N/n}$ satisfying $\|\mathbf{I}\|_\infty \leqslant B_I$ and $\|\mathbf{e}\|_\infty \leqslant B_e$ such that the following congruence holds:

$$\mathsf{Dec}_{s,\ell}(\mathbf{ct}) \equiv_{q_\ell} \Phi_{N\leftarrow n}\big(\Delta\mathsf{Lift}_n(\mathbf{a}) + \Delta t\mathbf{I} + \mathbf{e}\big).$$

Here and below, decryption congruences modulo q_ℓ are read in $\mathcal{R}_N$: the difference between the two sides lies in $q_\ell\mathcal{R}_N$, even though the message and error terms are expressed separately over the reals. We call such a pair $(\mathbf{I}, \mathbf{e})$ a *witness* to $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$. Since we have an equivalence modulo q_ℓ and not necessarily an equality, the decryption of $\mathbf{ct}$ may not yield $\mathbf{a}$. The following sufficient condition combines the lifting bound with the bounds on the packing isomorphisms above. It ensures that decryption avoids wraparound modulo q_ℓ and that rounding recovers the message.

Proposition 4 (Correct decryption). *Let* $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$. *Then* $(\mathsf{Dcd}_{\mathcal{A}_n} \circ \mathsf{Dec}_{s,\ell})(\mathbf{ct}) = \mathbf{a}$ *if the following two inequalities are satisfied:*

$$\Delta\|t\|_1\Big(\frac{1}{2} + B_I\Big) + B_e < \frac{q_\ell}{2\sqrt{n}}, \qquad B_e < \frac{\Delta}{2}.$$

Proof. Let $(\mathbf{I}, \mathbf{e})$ witness $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$. Set $\mathbf{Q} = \Delta\mathsf{Lift}_n(\mathbf{a}) + \Delta t\mathbf{I} + \mathbf{e}$. By (1), $\|\mathbf{Q}\|_\infty \leqslant \Delta\|t\|_1(1/2 + B_I) + B_e$. The hypotheses and Lemma 2 give $\|\Phi_{N\leftarrow n}(\mathbf{Q})\|_\infty \leqslant \sqrt{n}\,\|\mathbf{Q}\|_\infty < q_\ell/2$. It follows that $\mathsf{Dec}_{s,\ell}(\mathbf{ct}) = \Phi_{N\leftarrow n}\big(\Delta\mathsf{Lift}_n(\mathbf{a}) + \Delta t\mathbf{I} + \mathbf{e}\big)$ (that is, we have equality, and not only equivalence modulo q_ℓ). Now, $(\mathsf{Dcd}_{\mathcal{A}_n} \circ \mathsf{Dec}_{s,\ell})(\mathbf{ct}) = \mathbf{a}$ follows because $\|\mathbf{e}\|_\infty \leqslant B_e < \Delta/2$. □

Below, we therefore only need to provide bounds B_I and B_e for each ciphertext; decryption is then correct if the bounds from Proposition 4 are satisfied.

A.3 Fresh Encryption

For a fresh RLWE encryption error $e_{\mathsf{enc}} \in \mathcal{R}_N$, we assume the bound $\|\Phi_{n\leftarrow N}(e_{\mathsf{enc}})\|_\infty \leqslant B_{\mathsf{enc},n}$ in the lane representation. By Lemma 2, a native error bound $\|e_{\mathsf{enc}}\|_\infty \leqslant B_{\mathsf{enc}}$ allows $B_{\mathsf{enc},n} = (N/\sqrt{n})B_{\mathsf{enc}}$.

Proposition 5 (Fresh encryption). *Let* $\mathbf{a} \in \mathcal{A}_n^{N/n}$. *Then a ciphertext* $\mathbf{ct} \leftarrow \mathsf{Enc}_s\big(\lfloor \mathsf{Ecd}_{\mathcal{A}_n}(\mathbf{a})\rceil\big)$ *satisfies* $\mathbf{ct} \in \mathcal{C}_{s,L}(\mathbf{a}; B_I, B_e)$, *where:*

$$B_I = 0, \qquad B_e = B_{\mathsf{enc},n} + \frac{N}{2\sqrt{n}}.$$

Proof. Set $P = \mathsf{Ecd}_{\mathcal{A}_n}(\mathbf{a}) = \Delta\Phi_{N\leftarrow n}(\mathsf{Lift}_n(\mathbf{a}))$. Since $\lfloor P\rceil = P - [P]_1$, the encryption relation gives:

$$\mathsf{Dec}_{s,L}(\mathbf{ct}) \equiv_{q_L} P - [P]_1 + e_{\mathsf{enc}} = \Phi_{N\leftarrow n}\big(\Delta\mathsf{Lift}_n(\mathbf{a}) + \mathbf{e}\big),$$

where $\mathbf{e} = \Phi_{n\leftarrow N}(e_{\mathsf{enc}} - [P]_1)$. Thus, $(\mathbf{0}, \mathbf{e})$ witnesses the claimed membership. By the triangle inequality, Lemma 2, and the coefficientwise rounding bound $\|[P]_1\|_\infty \leqslant 1/2$:

$$\|\mathbf{e}\|_\infty \leqslant \|\Phi_{n\leftarrow N}(e_{\mathsf{enc}})\|_\infty + \|\Phi_{n\leftarrow N}([P]_1)\|_\infty \leqslant B_e.$$ □

A.4 Homomorphic Operations

For every key-switching operation (including relinearization) and rescaling operation considered below, we assume that the introduced errors $e_{\mathsf{ks}}, e_{\mathsf{rs}} \in \mathcal{S}_N$ satisfy:

$$\|e_{\mathsf{ks}}\|_\infty \leqslant B_{\mathsf{ks}}, \qquad \|\Phi_{n\leftarrow N}(e_{\mathsf{ks}})\|_\infty \leqslant B_{\mathsf{ks},n}, \qquad \|\Phi_{n\leftarrow N}(e_{\mathsf{rs}})\|_\infty \leqslant B_{\mathsf{rs},n}.$$

The bounds $B_{\mathsf{ks},n}$ and $B_{\mathsf{rs},n}$ refer to the lane representation in $\mathcal{S}_n^{N/n}$, while B_{ks} refers to the native ring $\mathcal{S}_N$. By Lemma 2, a native error bound $B_\star$ allows $B_{\star,n} = (N/\sqrt{n})B_\star$ for $\star \in \{\mathsf{ks}, \mathsf{rs}\}$.

Proposition 6 (Plaintext–ciphertext addition). *Consider a ciphertext* $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$ *and a plaintext* $\mathbf{a}' \in \mathcal{A}_n^{N/n}$. *Set* $P = \Delta\Phi_{N\leftarrow n}(\mathsf{Lift}_n(\mathbf{a}'))$. *Then* $\mathbf{ct}_{\mathsf{add}} \in \mathcal{C}_{s,\ell}(\mathbf{a} + \mathbf{a}'; B_{I,\mathsf{add}}, B_{e,\mathsf{add}})$, *where* $\mathbf{ct}_{\mathsf{add}} \leftarrow \mathsf{Add}(\lfloor P\rceil, \mathbf{ct})$ *and:*

$$B_{I,\mathsf{add}} = B_I + 1, \qquad B_{e,\mathsf{add}} = B_e + \frac{N}{2\sqrt{n}}.$$

Proof. Let $(\mathbf{I}, \mathbf{e})$ witness $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$. By Section 4.1, $\mathsf{Lift}_n(\mathbf{a}) + \mathsf{Lift}_n(\mathbf{a}') = \mathsf{Lift}_n(\mathbf{a}+\mathbf{a}') + t\mathbf{J}$ for some $\mathbf{J} \in \mathcal{R}_n^{N/n}$. Moreover, by isolating $\mathbf{J}$, (1) gives $\|\mathbf{J}\|_\infty \leqslant 3/2$, and hence $\|\mathbf{J}\|_\infty \leqslant 1$ since $\mathbf{J}$ has integer coefficients. Setting $\mathbf{I}_{\mathsf{add}} = \mathbf{I} + \mathbf{J}$ and $\mathbf{e}_{\mathsf{add}} = \mathbf{e} - \Phi_{n\leftarrow N}([P]_1)$, and using $\lfloor P \rceil = P - [P]_1$, we obtain:

$$\mathsf{Dec}_{s,\ell}(\mathbf{ct}_{\mathsf{add}}) \equiv_{q_\ell} \Phi_{N\leftarrow n}\big(\Delta \mathsf{Lift}_n(\mathbf{a}+\mathbf{a}') + \Delta t \mathbf{I}_{\mathsf{add}} + \mathbf{e}_{\mathsf{add}}\big).$$

Finally, $\|\mathbf{I}_{\mathsf{add}}\|_\infty \leqslant B_I + 1$. By Lemma 2 and $\|[P]_1\|_\infty \leqslant 1/2$, we have $\|\mathbf{e}_{\mathsf{add}}\|_\infty \leqslant B_e + N/(2\sqrt{n})$, proving the claim. □

Proposition 7 (Ciphertext–ciphertext addition). *Consider ciphertexts* $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$ *and* $\mathbf{ct}' \in \mathcal{C}_{s,\ell}(\mathbf{a}'; B'_I, B'_e)$. *Then* $\mathbf{ct}_{\mathsf{add}} \in \mathcal{C}_{s,\ell}(\mathbf{a}+\mathbf{a}'; B_{I,\mathsf{add}}, B_{e,\mathsf{add}})$, *where* $\mathbf{ct}_{\mathsf{add}} \leftarrow \mathsf{Add}(\mathbf{ct}, \mathbf{ct}')$ *and:*

$$B_{I,\mathsf{add}} = B_I + B'_I + 1, \qquad B_{e,\mathsf{add}} = B_e + B'_e.$$

Proof. Let $(\mathbf{I}, \mathbf{e})$ and $(\mathbf{I}', \mathbf{e}')$ witness that $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$ and $\mathbf{ct}' \in \mathcal{C}_{s,\ell}(\mathbf{a}'; B'_I, B'_e)$, respectively. We can again write $\mathsf{Lift}_n(\mathbf{a}) + \mathsf{Lift}_n(\mathbf{a}') = \mathsf{Lift}_n(\mathbf{a}+\mathbf{a}') + t\mathbf{J}$ for some $\mathbf{J} \in \mathcal{R}_n^{N/n}$ with $\|\mathbf{J}\|_\infty \leqslant 1$. Setting $\mathbf{I}_{\mathsf{add}} = \mathbf{I} + \mathbf{I}' + \mathbf{J}$ and $\mathbf{e}_{\mathsf{add}} = \mathbf{e} + \mathbf{e}'$, we obtain:

$$\mathsf{Dec}_{s,\ell}(\mathbf{ct}_{\mathsf{add}}) \equiv_{q_\ell} \Phi_{N\leftarrow n}\big(\Delta \mathsf{Lift}_n(\mathbf{a}+\mathbf{a}') + \Delta t(\mathbf{I} + \mathbf{I}' + \mathbf{J}) + \mathbf{e} + \mathbf{e}'\big).$$

Finally, $\|\mathbf{I}_{\mathsf{add}}\|_\infty \leqslant B_I + B'_I + 1$ and $\|\mathbf{e}_{\mathsf{add}}\|_\infty \leqslant B_e + B'_e$, proving the claim. □

Proposition 8 (Plaintext–ciphertext multiplication). *For a level* $\ell \in [\![1, L]\!]$, *consider a ciphertext* $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$ *and a plaintext* $\mathbf{a}' \in \mathcal{A}_n^{N/n}$. *Set* $P = \Delta \Phi_{N\leftarrow n}(\mathsf{Lift}_n(\mathbf{a}'))$. *Then* $\mathbf{ct}_{\mathsf{mul}} \in \mathcal{C}_{s,\ell-1}(\mathbf{a} \odot \mathbf{a}'; B_{I,\mathsf{mul}}, B_{e,\mathsf{mul}})$, *where* $\mathbf{ct}_{\mathsf{mul}} \leftarrow \mathsf{Rescale}(\mathsf{Mul}(\lfloor P \rceil, \mathbf{ct}))$ *and:*

$$B_{I,\mathsf{mul}} = \frac{n\|t\|_1}{2}\Big(\frac{1}{2} + B_I\Big) + \frac{1}{2},$$
$$B_{e,\mathsf{mul}} = \frac{N\sqrt{n}\|t\|_1}{2}\Big(\frac{1}{2} + B_I\Big) + \frac{n\|t\|_1}{2}B_e + \frac{N\sqrt{n}B_e}{2\Delta} + B_{\mathsf{rs},n}.$$

Proof. The proof of Proposition 9 below applies with $\mathbf{I}' = \mathbf{0}$ and $\mathbf{e}' = -\Phi_{n\leftarrow N}([P]_1)$ with $\|\mathbf{e}'\|_\infty \leqslant N/(2\sqrt{n})$. Plaintext–ciphertext multiplication requires no relinearization. Thus, setting $B'_I = 0$ and $B'_e = N/(2\sqrt{n})$, and omitting the key-switching error term, gives the claimed overflow and error bounds. □

Proposition 9 (Ciphertext–ciphertext multiplication). *For a level* $\ell \in [\![1, L]\!]$, *consider ciphertexts* $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$ *and* $\mathbf{ct}' \in \mathcal{C}_{s,\ell}(\mathbf{a}'; B'_I, B'_e)$. *Then* $\mathbf{ct}_{\mathsf{mul}} \in \mathcal{C}_{s,\ell-1}(\mathbf{a} \odot \mathbf{a}'; B_{I,\mathsf{mul}}, B_{e,\mathsf{mul}})$, *where* $\mathbf{ct}_{\mathsf{mul}} \leftarrow \mathsf{Rescale}(\mathsf{Mul}(\mathbf{ct}, \mathbf{ct}'))$ *and:*

$$B_{I,\mathsf{mul}} = n\|t\|_1\Big(\frac{1}{2} + B_I\Big)\Big(\frac{1}{2} + B'_I\Big) + \frac{1}{2},$$
$$B_{e,\mathsf{mul}} = n\|t\|_1\Big(\frac{1}{2} + B_I\Big)B'_e + n\|t\|_1\Big(\frac{1}{2} + B'_I\Big)B_e + \frac{nB_eB'_e + B_{\mathsf{ks},n}}{\Delta} + B_{\mathsf{rs},n}.$$

Proof. Let $(\mathbf{I}, \mathbf{e})$ and $(\mathbf{I}', \mathbf{e}')$ witness $\mathbf{ct} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$ and $\mathbf{ct}' \in \mathcal{C}_{s,\ell}(\mathbf{a}'; B'_I, B'_e)$, respectively. Set $\mathbf{Q} = \mathsf{Lift}_n(\mathbf{a}) + t\mathbf{I}$ and $\mathbf{Q}' = \mathsf{Lift}_n(\mathbf{a}') + t\mathbf{I}'$. By Section 4.1, we have $\mathbf{Q} \odot \mathbf{Q}' = \mathsf{Lift}_n(\mathbf{a} \odot \mathbf{a}') + t\mathbf{I}_{\mathsf{mul}}$ for some $\mathbf{I}_{\mathsf{mul}} \in \mathcal{R}_n^{N/n}$. Isolating $\mathbf{I}_{\mathsf{mul}}$ gives:

$$\mathbf{I}_{\mathsf{mul}} = \Big(\frac{\mathsf{Lift}_n(\mathbf{a})}{t} + \mathbf{I}\Big) \odot \mathbf{Q}' - \frac{\mathsf{Lift}_n(\mathbf{a} \odot \mathbf{a}')}{t}.$$

Using (1) and the coefficient bound $\|FG\|_\infty \leqslant n\|F\|_\infty\|G\|_\infty$ for all $F, G \in \mathcal{S}_n$, we obtain $\|\mathbf{I}_{\mathsf{mul}}\|_\infty \leqslant B_{I,\mathsf{mul}}$. Since $\Phi_{N\leftarrow n}$ is a ring isomorphism, multiplication with relinearization followed by rescaling gives:

$$\mathsf{Dec}_{s,\ell-1}(\mathbf{ct}_{\mathsf{mul}}) \equiv_{q_{\ell-1}} \Phi_{N\leftarrow n}\big(\Delta\mathsf{Lift}_n(\mathbf{a}\odot\mathbf{a}') + \Delta t\mathbf{I}_{\mathsf{mul}} + \mathbf{e}_{\mathsf{mul}}\big),$$

where $\mathbf{e}_{\mathsf{mul}} = \mathbf{Q}\odot\mathbf{e}' + \mathbf{Q}'\odot\mathbf{e} + (\mathbf{e}\odot\mathbf{e}' + \Phi_{n\leftarrow N}(e_{\mathsf{ks}}))/\Delta + \Phi_{n\leftarrow N}(e_{\mathsf{rs}})$. Finally, using $\|\mathbf{Q}\|_\infty \leqslant \|t\|_1(1/2 + B_I)$ and $\|\mathbf{Q}'\|_\infty \leqslant \|t\|_1(1/2 + B_I')$, we obtain:

$$\|\mathbf{e}_{\mathsf{mul}}\|_\infty \leqslant n\|t\|_1\Big(\frac{1}{2} + B_I\Big)B_e' + n\|t\|_1\Big(\frac{1}{2} + B_I'\Big)B_e + \frac{nB_eB_e' + B_{\mathsf{ks},n}}{\Delta} + B_{\mathsf{rs},n} = B_{e,\mathsf{mul}}. \qquad \square$$

A.5 Lazy Flattening

We use the matrices and level costs from Section 6.1. For the inputs considered, suppose the forward $\mathsf{LinTrans}$ call with $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{V}_{N\leftarrow n})$ in line 1 of Algorithm 1 introduces $e_{N\leftarrow n} \in \mathcal{S}_N$ with $\|e_{N\leftarrow n}\|_\infty \leqslant B_{N\leftarrow n}$, and the reverse $\mathsf{LinTrans}$ call with $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{V}_{n\leftarrow N})$ in line 5 introduces $\mathbf{e}_{n\leftarrow N} \in \mathcal{S}_n^{N/n}$ with $\|\mathbf{e}_{n\leftarrow N}\|_\infty \leqslant B_{n\leftarrow N}$. The forward call divides each lane by t, collects the coefficients in $\mathcal{S}_N$, and changes scale from Δ to q_0. The reverse call returns the coefficients to $\mathcal{S}_n^{N/n}$, multiplies each lane by t, and restores scale Δ. These are additive errors in the respective output representations. The coefficient maps κ preserve coefficient norms for every n. For related error estimates, see [KPK+22].

Proposition 10 (Correctness of lazy flattening). *Set $\ell = \ell_{N\leftarrow n}$ and $\tilde{\ell} = L - \ell_{n\leftarrow N}$, and suppose $\tilde{\ell} \geqslant 0$. Consider $\mathbf{ct}_{\mathsf{in}} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$. Then Algorithm 1 returns $\mathbf{ct}_{\mathsf{out}} \in \mathcal{C}_{s,\tilde{\ell}}(\mathbf{a}; \tilde{B}_I, \tilde{B}_e)$, where:*

$$\tilde{B}_I = \frac{h'}{2} + 1 + \frac{\big\|\frac{1}{t}\big\|_1 B_e}{\Delta} + \frac{B_{N\leftarrow n} + B_{\mathsf{ks}}}{q_0},$$
$$\tilde{B}_e = B_e + \frac{\Delta\|t\|_1}{q_0}\big(B_{N\leftarrow n} + 2B_{\mathsf{ks}}\big) + B_{n\leftarrow N}.$$

Proof. Let $(\mathbf{I}, \mathbf{e})$ witness the input membership. After line 2, the ciphertext decrypts (modulo q_0) to $q_0\kappa_{N\leftarrow n}(\mathsf{Lift}_n(\mathbf{a})/t) + e_{\mathsf{pre}}$, where e_{pre} collects the errors before modulus raising:

$$e_{\mathsf{pre}} = \frac{q_0}{\Delta}\kappa_{N\leftarrow n}\Big(\frac{\mathbf{e}}{t}\Big) + e_{N\leftarrow n} + e_{\mathsf{ks}};$$

the contribution $q_0\kappa_{N\leftarrow n}(\mathbf{I})$ vanishes modulo q_0. Writing this ciphertext as $\mathbf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1) \in \mathcal{R}_{N,q_0}^2$ with centered coefficients modulo q_0, let $m_{\mathsf{mr}} = \mathsf{ct}_0 s' + \mathsf{ct}_1$ be its unreduced decryption polynomial. It satisfies $\|m_{\mathsf{mr}}\|_\infty \leqslant q_0(h'+1)/2$, and for some $\tilde{\mathbf{I}} \in \mathcal{R}_n^{N/n}$ we have:

$$m_{\mathsf{mr}} = q_0\kappa_{N\leftarrow n}\Big(\frac{\mathsf{Lift}_n(\mathbf{a})}{t}\Big) + e_{\mathsf{pre}} + q_0\kappa_{N\leftarrow n}(\tilde{\mathbf{I}}).$$

Here $\tilde{\mathbf{I}}$ is the output overflow. By (1), the error bounds, and coefficient-norm preservation under $\kappa_{N\leftarrow n}$:

$$\|\tilde{\mathbf{I}}\|_\infty \leqslant \frac{h'+2}{2} + \frac{\|e_{\mathsf{pre}}\|_\infty}{q_0} \leqslant \frac{h'}{2} + 1 + \frac{\big\|\frac{1}{t}\big\|_1 B_e}{\Delta} + \frac{B_{N\leftarrow n} + B_{\mathsf{ks}}}{q_0} = \tilde{B}_I.$$

Line 3 preserves m_{mr} modulo q_L, while line 4 adds e'_{ks}. At output level $\tilde{\ell}$, the reverse transformation in line 5 gives:

$$\mathsf{Dec}_{s,\tilde{\ell}}(\mathbf{ct}_{\mathsf{out}}) \equiv_{q_{\tilde{\ell}}} \Phi_{N\leftarrow n}\big(\Delta\mathsf{Lift}_n(\mathbf{a}) + \Delta t\tilde{\mathbf{I}} + \tilde{\mathbf{e}}\big),$$

where:

$$\tilde{\mathbf{e}} = \mathbf{e} + \frac{\Delta}{q_0} t\, \kappa_{n\leftarrow N}\big(e_{N\leftarrow n} + e_{\mathsf{ks}} + e'_{\mathsf{ks}}\big) + \mathbf{e}_{n\leftarrow N}.$$

Finally, $\kappa_{n\leftarrow N}$ preserves coefficient norms and $\|tF\|_\infty \leqslant \|t\|_1 \|F\|_\infty$ for every $F \in \mathcal{S}_n$. Applying this inequality in each lane gives $\|\tilde{\mathbf{e}}\|_\infty \leqslant \tilde{B}_e$, as required. □

Remark 3. In particular, if $\|1/t\|_1 B_e/\Delta + (B_{N\leftarrow n} + B_{\mathsf{ks}})/q_0 < 1/2$, the integer coefficients of $\tilde{\mathbf{I}}$ give $\|\tilde{\mathbf{I}}\|_\infty \leqslant h'/2+1$. Applying Proposition 4 with the output bounds and level gives sufficient conditions for correct decoding.

A.6 Arithmetic Bootstrapping

We use the matrices and level costs from Section 6.2. For the inputs considered, suppose the forward and reverse $\mathsf{LinTrans}$ calls with $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{V}_{N\leftarrow n})$ and $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{W}_{n\leftarrow N})$ in line 1 and line 12 of Algorithm 2 introduce $e_{N\leftarrow n} \in \mathcal{S}_N$ and $\mathbf{e}_{n\leftarrow N} \in \mathcal{S}_n^{N/n}$, with infinity norms bounded by $B_{N\leftarrow n}$ and $B_{n\leftarrow N}$, respectively. In the error branch, the calls with $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{W}_{2\leftarrow N})$ and $\mathsf{Ecd}_{\mathbb{C}}(\mathbf{U}_{n\leftarrow 2})$ in line 8 and line 10 introduce $\mathbf{e}_{2\leftarrow N} \in \mathcal{S}_2^{N/2}$ and $\mathbf{e}_{n\leftarrow 2} \in \mathcal{S}_n^{N/n}$, with infinity norms bounded by $B_{2\leftarrow N}$ and $B_{n\leftarrow 2}$, respectively. These are additive errors in the respective output representations. The coefficient maps κ preserve these norms. For an input tolerance $0 \leqslant \varepsilon_{\mathsf{em}} < 1/2$, assume that the $\mathsf{EvalMod}$ call in line 9 maps each real and imaginary component $x+e$, where $x \in \mathbb{Z}$ with $|x| \leqslant h'/2+1$ and $|e| \leqslant \varepsilon_{\mathsf{em}}$, to $e + e_{\mathsf{em}}$ with $|e_{\mathsf{em}}| \leqslant B_{\mathsf{em}}/\Delta$. The error e_{em} includes approximation and homomorphic evaluation errors.

Proposition 11 (Correctness of arithmetic bootstrapping). *Set $\ell = \ell_{N\leftarrow n}$ and $\tilde{\ell} = L - \ell_{2\leftarrow N} - \ell_{\mathsf{em}} - \ell_{n\leftarrow 2}$, and suppose $\tilde{\ell} \geqslant 0$ and $L \geqslant \tilde{\ell} + \ell_{n\leftarrow N}$. Consider $\mathbf{ct}_{\mathsf{in}} \in \mathcal{C}_{s,\ell}(\mathbf{a}; B_I, B_e)$. Assume the following condition:*

$$\frac{B_e + B_{2\leftarrow N}}{\Delta} + \frac{\|t\|_1 (B_{N\leftarrow n} + B_{\mathsf{ks}}) + B_{\mathsf{ks}}}{q_0} \leqslant \varepsilon_{\mathsf{em}}. \tag{3}$$

Then Algorithm 2 returns $\mathbf{ct}_{\mathsf{out}} \in \mathcal{C}_{s,\tilde{\ell}}(\mathbf{a}; \tilde{B}_I, \tilde{B}_e)$, where:

$$\tilde{B}_I = \frac{h'}{2} + 1 + \frac{\left\|\frac{1}{t}\right\|_1 B_e}{\Delta} + \frac{B_{N\leftarrow n} + B_{\mathsf{ks}}}{q_0},$$
$$\tilde{B}_e = B_{n\leftarrow N} + B_{2\leftarrow N} + B_{\mathsf{em}} + B_{n\leftarrow 2} + \frac{2\Delta}{q_0} B_{\mathsf{ks}}.$$

Proof. Let $(\mathbf{I}, \mathbf{e})$ witness the input membership. Up to line 3, the algorithm follows lazy flattening with top level $\tilde{\ell} + \ell_{n\leftarrow N}$. We use e_{pre} and $\tilde{\mathbf{I}}$ from its proof, where e_{ks} is the error of the initial key switch to s' and $\|\tilde{\mathbf{I}}\|_\infty \leqslant \tilde{B}_I$. After the exact multiplication in line 4, the ciphertext under s' decrypts modulo $q_{\tilde{\ell}+\ell_{n\leftarrow N}}$ to $q_0 \kappa_{N\leftarrow n}(\mathsf{Lift}_n(\mathbf{a}) + t\tilde{\mathbf{I}}) + e'_{\mathsf{pre}}$, where:

$$e'_{\mathsf{pre}} = t\big(X^{N/n}\big) e_{\mathsf{pre}} = \frac{q_0}{\Delta} \kappa_{N\leftarrow n}(\mathbf{e}) + t\big(X^{N/n}\big)\big(e_{N\leftarrow n} + e_{\mathsf{ks}}\big).$$

Here we use $\kappa_{N\leftarrow n}(t\mathbf{F}) = t\big(X^{N/n}\big)\kappa_{N\leftarrow n}(\mathbf{F})$ for $\mathbf{F} \in \mathcal{S}_n^{N/n}$. Line 5 saves the error branch under s' and removes the integer multiple of q_0. Writing this ciphertext as $\mathbf{ct}' = (\mathsf{ct}'_0, \mathsf{ct}'_1) \in \mathcal{R}^2_{N,q_0}$, let

$m'_{\mathsf{mr}} = \mathsf{ct}'_0 s' + \mathsf{ct}'_1$ be its unreduced decryption polynomial. It satisfies $\|m'_{\mathsf{mr}}\|_\infty \leqslant q_0(h'+1)/2$, and for some $J \in \mathcal{R}_N$ we have $m'_{\mathsf{mr}} = e'_{\mathsf{pre}} + q_0 J$. By (3), $\|e'_{\mathsf{pre}}\|_\infty / q_0 \leqslant \varepsilon_{\mathsf{em}} < 1/2$, so:

$$\|J\|_\infty \leqslant \frac{h'+1}{2} + \frac{\|e'_{\mathsf{pre}}\|_\infty}{q_0} < \frac{h'}{2} + 1.$$

Line 6 preserves m'_{mr} modulo q_L, while line 7 adds e'_{ks} with $\|e'_{\mathsf{ks}}\|_\infty \leqslant B_{\mathsf{ks}}$. Line 8 exposes the coefficients in $\mathcal{S}_2^{N/2}$ at scale Δ. Condition (3) ensures that each normalized coefficient differs from the corresponding integer coefficient of $\kappa_{2\leftarrow N}(J)$ by at most $\varepsilon_{\mathsf{em}}$, including the transformation error. Thus all real and imaginary components lie in the prescribed domain, so line 9 removes the integer contribution with additional error bounded by B_{em}. After line 10, the error branch represents:

$$\frac{\Delta}{q_0}\kappa_{n\leftarrow N}(e'_{\mathsf{pre}} + e'_{\mathsf{ks}}) + \mathbf{e}_{\mathsf{ext}}, \qquad \|\mathbf{e}_{\mathsf{ext}}\|_\infty \leqslant B_{2\leftarrow N} + B_{\mathsf{em}} + B_{n\leftarrow 2}.$$

The errors add without norm amplification because $\kappa_{n\leftarrow 2}$ only permutes coefficients. The retained branch is then switched to s in line 11, introducing e''_{ks} with $\|e''_{\mathsf{ks}}\|_\infty \leqslant B_{\mathsf{ks}}$. At output level $\tilde{\ell}$, the reverse transformation in line 12 gives the lane representative:

$$\Delta\mathsf{Lift}_n(\mathbf{a}) + \Delta t\tilde{\mathbf{I}} + \frac{\Delta}{q_0}\kappa_{n\leftarrow N}(e'_{\mathsf{pre}} + e''_{\mathsf{ks}}) + \mathbf{e}_{n\leftarrow N}.$$

Subtraction in line 13 cancels the contribution of e'_{pre}, giving:

$$\mathsf{Dec}_{s,\tilde{\ell}}(\mathbf{ct}_{\mathsf{out}}) \equiv_{q_{\tilde{\ell}}} \Phi_{N\leftarrow n}\big(\Delta\mathsf{Lift}_n(\mathbf{a}) + \Delta t\tilde{\mathbf{I}} + \tilde{\mathbf{e}}\big),$$

where:

$$\tilde{\mathbf{e}} = \mathbf{e}_{n\leftarrow N} - \mathbf{e}_{\mathsf{ext}} + \frac{\Delta}{q_0}\kappa_{n\leftarrow N}(e''_{\mathsf{ks}} - e'_{\mathsf{ks}}).$$

Finally, coefficient-norm preservation under $\kappa_{n\leftarrow N}$ gives $\|\tilde{\mathbf{e}}\|_\infty \leqslant \tilde{B}_e$. The level assumptions allow both branches to be reduced to $q_{\tilde{\ell}}$, proving the claimed output membership. □

A.7 Final Addition of Arithmetic-to-Digit Conversion

We provide further details on the correctness of Algorithm 3. Recall that $M = \beta^n + 1$. Here we identify the message $a \in \mathbb{Z}_M$ with its unique integer representative in $[\![M]\!]$. The sum $\mathbf{v} = (v_i)_{i\in[\![n]\!]}$ of the two input digit vectors has entries in $[\![0, 2\beta-2]\!]$ and satisfies $\sum_{i=0}^{n-1} v_i\beta^i \equiv_M a - 1$. We write $\mathbf{b}_0 = (1, 0, \ldots, 0)$ for the vector supported at the initial digit.

Denote the three symbol vectors before the loop by $\boldsymbol{\sigma}_{\mathsf{in}}$ and $\boldsymbol{\sigma}_{\mathsf{in}}^{(b)}$ for $b \in \{0,1\}$, and the corresponding vectors after it by $\boldsymbol{\sigma}_{\mathsf{out}}$ and $\boldsymbol{\sigma}_{\mathsf{out}}^{(b)}$. After the loop, the entry at position i describes the carry behavior of all n digits, starting with a carry entering position $i+1$ and proceeding cyclically through position i, with indices taken modulo n. When $\mathsf{MergeCarry}$ combines two consecutive blocks, it returns the symbol of the second block if that symbol is 0 or 2, and the symbol of the first block if the second has symbol 1. Repeated merging therefore returns the last 0 or 2 encountered in the traversal order: any following 1s preserve that symbol. Consequently, each block of 1s is filled with the nearest preceding symbol different from 1, looking backward cyclically in the corresponding input vector. If every input symbol is 1, every output symbol remains 1.

Suppose first that $\boldsymbol{\sigma}_{\mathsf{in}} \neq \mathbf{1}$. Let $s \in \{0, 2\}$ be its last symbol different from 1. The initial block of 1s, which may be empty, is therefore filled with s in $\boldsymbol{\sigma}_{\mathsf{out}}$. The modification in line 3 replaces an initial 1 by $2b$ and leaves an initial 0 or 2 unchanged. Consequently, the initial block of 1s in $\boldsymbol{\sigma}_{\mathsf{in}}$ is filled with $2b$ in $\boldsymbol{\sigma}^{(b)}_{\mathsf{out}}$. Outside this initial block, all three output vectors agree: the first original symbol different from 1 fixes its outgoing carry independently of the incoming carry. Applying $\mathsf{EvalCarry}(\mathbf{0}, \cdot)$ turns the initial blocks of $\boldsymbol{\sigma}^{(0)}_{\mathsf{out}}$, $\boldsymbol{\sigma}^{(1)}_{\mathsf{out}}$, and $\boldsymbol{\sigma}_{\mathsf{out}}$ into carries 0, 1, and $r = s/2$, respectively, where r is the carry leaving the last digit. Thus the combination in line 10 gives $0 + 1 - r = 1 - r$ on the initial block, and preserves the common carry u elsewhere because $u + u - u = u$. These are exactly the carries obtained by feeding the complemented final carry $1 - r$ into the initial digit. In Figure 5, for example, the three carry rows start with 000, 111, and 111, producing 000 in $\mathbf{c}$.

Finally, align_1 shifts the carries to the positions receiving them, but its initial entry is $\beta - 1 - r$. Subtracting $(\beta - 2)\mathbf{b}_0$ changes this entry to $1 - r$, as required. Let c_i, d_i denote the entries at position i of $\mathbf{c}, \mathbf{d}$, respectively, and set $c_{-1} = 1 - c_{n-1}$. Then line 12 gives $d_i = v_i + c_{i-1} - \beta c_i$ for $i \in [\![n]\!]$. Since $c_i = \lfloor (v_i + c_{i-1})/\beta \rfloor$, each digit lies in $[\![\beta]\!]$. The complemented wraparound supplies the missing $+1$ modulo M:

$$\sum_{i=0}^{n-1} d_i \beta^i = \sum_{i=0}^{n-1} v_i \beta^i + c_{-1} - \beta^n c_{n-1} = \sum_{i=0}^{n-1} v_i \beta^i + 1 - M c_{n-1} \equiv_M a.$$

The digit sum and a both lie in $[\![M]\!]$, so they are equal.

If $\boldsymbol{\sigma}_{\mathsf{in}} = \mathbf{1}$, then every $v_i = \beta - 1$ and $a = \beta^n$. The loop gives $\boldsymbol{\sigma}_{\mathsf{out}} = \mathbf{1}$ and $\boldsymbol{\sigma}^{(b)}_{\mathsf{out}} = 2b \cdot \mathbf{1}$, so the carry combination yields $\mathbf{c} = \mathbf{1}$. The output formula then gives $d_0 = -1$ and $d_i = 0$ for $i \in [\![1, n-1]\!]$, returning the exceptional representation $-\mathbf{b}_0$.

B Implementation Details and Additional Experiments

We supplement Section 8 with the implemented level schedules, scale adjustments, operation counts, and further precision and timing measurements.

B.1 Further Implementation Parameters

Modulus Chains. We use the residue number system (RNS) variant of CKKS [CHK+19] and retain $q_0, \ldots, q_L$ for the ciphertext moduli. The successive ratios $q_{\ell+1}/q_\ell$ are distinct primes, replacing the fixed factor Δ in Section 3. For arithmetic and refreshing, Table 8 gives the selected parameters. Starting from the bottom, the arithmetic chain contains a 60-bit prime q_0, then $\ell_{N \leftarrow n}$ primes of 56 bits, one 60-bit prime used for multiplication, $\max(\ell_{n \leftarrow N}, \ell_{n \leftarrow 2})$ primes of 56 bits, eight 60-bit primes for $\mathsf{EvalMod}$, and three 56-bit primes for $\mathbf{W}_{2 \leftarrow N}$.

The conversion chain begins with q_0 and the next $\ell_{N \leftarrow n}$ primes of the arithmetic chain. We then append 27 to 30 primes: 50-bit primes for $n \leqslant 8\,192$, 48-bit primes for $n = 2^{14}$, and 46-bit primes for $n = 2^{15}$. The corresponding target scales are $\Delta = 2^{50}$, 2^{48}, and 2^{46}. Including the auxiliary primes of Section 8.1, $\log(q_L P_{\mathsf{aux}})$ ranges from 1 716 to 1 742.

The shared primes give identical ciphertext moduli up to level $\ell_{N \leftarrow n}$. Since both chains also use the same main secret s, a ciphertext at any of these levels can be used unchanged with either parameter set. Each chain has its own evaluation keys and an ephemeral secret of Hamming weight $h' = 32$.

In a test at $n = 64$, we lowered the scale for homomorphic arithmetic from $\Delta = 2^{60}$ to $\Delta = 2^{50}$ and used 50-bit ciphertext-modulus primes, keeping q_0 unchanged. The logarithm of the normalized error after arithmetic bootstrapping increased from -35 to -21, allowing only two multiplications between error refreshings instead of three.

Table 8. Refreshing parameters at $N = 2^{16}$ and $t = X - 2$. The transformation depths minimize the measured runtime over the tested splits. All rows use $\ell_{\mathsf{em}} = 8$ and $\ell_{2\leftarrow N} = 3$. The modulus sizes $\log(q_L P_{\mathsf{aux}})$ remain below the 1 747-bit limit for 128-bit security with a uniform ternary main secret [BCC+25].

$\log n$	$\ell_{N\leftarrow n}$	$\ell_{n\leftarrow N}$	$\ell_{n\leftarrow 2}$	L	$\log(q_L P_{\mathsf{aux}})$	Galois keys
3	2	2	1	16	1 175	66
4	2	2	1	16	1 175	58
5	2	2	1	16	1 175	52
6	2	2	2	16	1 175	40
7	2	2	2	16	1 175	47
8	2	2	2	16	1 175	49
9	2	2	2	16	1 175	50
10	2	2	2	16	1 175	46
11	1	1	2	15	1 119	39
12	1	1	2	15	1 119	52
14	1	1	2	15	1 119	64

Scale Considerations. In this RNS implementation, rescaling from level $\ell + 1$ to ℓ divides by the prime $q_{\ell+1}/q_\ell$, which need not equal Δ. Multiplication at scale Δ followed by rescaling therefore gives scale $\Delta' = \Delta^2 q_\ell / q_{\ell+1}$. The forward transformation must use this actual input scale. We therefore encode the matrix $(\Delta/\Delta') \cdot \mathbf{V}_{N\leftarrow n}$ instead of $\mathbf{V}_{N\leftarrow n}$. Without this adjustment, reduction modulo q_0 leaves the overflow contribution $q_0(\Delta'/\Delta - 1)\kappa_{N\leftarrow n}(\mathbf{I})$; after one multiplication, its normalized error has logarithm about -39 in the arithmetic benchmarks. We cache the encodings for the few input scales encountered in the evaluation loop. Lazy flattening and arithmetic bootstrapping return their target scale Δ and otherwise follow Section 6; multiplication by $t(X^{N/n})$ uses an integer plaintext at scale one and consumes no level.

EvalMod and Its Input Range. For arithmetic bootstrapping, we apply Lattigo's `mod1` circuit separately to real and imaginary slot components, using the approximation parameters of Section 8.1. Its input and output normalization factors are absorbed into $\mathbf{W}_{2\leftarrow N}$ and $\mathbf{U}_{n\leftarrow 2}$. The discrete cosine approximation covers neighborhoods of the integers from -15 to 15; accuracy decreases near the endpoints. To estimate the probability of an overflow outside this range, write $m'_{\mathsf{mr}}/q_0 = J + e'_{\mathsf{pre}}/q_0$ as in the proof of Proposition 11. Following [BCC+25, Appendix A], we model each coefficient of m'_{mr}/q_0 as a sum of $h' + 1 = 33$ independent uniform variables on $(-1/2, 1/2]$. Assuming $\|e'_{\mathsf{pre}}\|_\infty / q_0 < 0.1$, an integer overflow of magnitude at least 16 requires the corresponding sum to have magnitude greater than $16 - 0.1 = 15.9$. The sum has maximum 16.5, so each tail beyond 15.9 has probability $(16.5 - 15.9)^{33}/33! = 0.6^{33}/33!$. A union bound over both tails and the $N = 2^{16}$ coefficients gives $\Pr\left[\|J\|_\infty \geqslant 16\right] \leqslant 2N \cdot 0.6^{33}/33! < 2^{-130}$. This estimate relies on the stated distributional model and error assumption. In our experiments, observed overflows have magnitude about 7.

B.2 Refreshing Precision

At $N = 2^{16}$, the logarithm of the normalized error after arithmetic bootstrapping stays near -35 across the tested input errors, scales, and repeated calls. Using a sparse main secret of Hamming

weight 192 gives about -37; the corresponding values are about -23 for our GZ port and -24.4 in [GZ26, Table 3].

Table 9 varies N while fixing $t = X - 2$, the modulus chain, scale, ephemeral-secret weight, and EvalMod parameters. It compares four lanes with fixed lane degree $n = 256$, listing their common case only once. With four lanes, each doubling of N increases the logarithmic error by about 1.1 bits; at the other shared dimensions, the two lane configurations differ by at most 0.3 bits.

Table 9. Base-two logarithm of the normalized error after arithmetic bootstrapping with $t = X - 2$. The modulus chain and other parameters are fixed; configurations with $N < 2^{16}$ are insecure.

$\log N$	$\log n$	$\log(\|\tilde{\mathbf{e}}\|_\infty/\Delta)$
10	8	-41.7
11	9	-40.5
11	8	-40.3
12	10	-39.5
12	8	-39.4
13	11	-38.2
13	8	-38.2
14	12	-37.4
14	8	-37.1
15	13	-36.3
16	14	-35.0

Table 10 lists the arithmetic bootstrapping times behind the radix experiment of Section 8.2, with $M = \beta^n + 1$. Larger β enlarges the modulus at fixed n without increasing the refreshing time.

Table 10. Arithmetic bootstrapping times on AWS at $N = 2^{16}$ for $t = X - \beta$, with fixed EvalMod parameters. Times are means of ten iterations from the profiling run of Table 12.

β	$\|t\|_1$	$n = 8$		$n = 4\,096$	
		Bit length of M	ArithBoot	Bit length of M	ArithBoot
2	3	9	7.96 s	4 097	6.42 s
8	9	25	7.90 s	12 289	6.41 s
26	27	38	7.91 s	19 254	6.36 s
64	65	49	7.80 s	24 577	6.34 s

B.3 Operation Counts and Timings

Table 11 compares operation counts and Galois keys for both refreshing procedures. Table 12 gives step timings on AWS and on the MacBook Pro specified in Supplementary Material B.5; its matrix rows denote the corresponding LinTrans calls. The ModRaise row includes both modulus raisings and all three key switches. The AWS columns provide the data grouped in Figure 6.

B.4 Arithmetic-to-Digit Conversion

Level Schedule. Using the parameters of Supplementary Material B.1, the implemented schedule for $n = 64$ is as follows:

1. Multiply on the arithmetic chain and rescale to level 2. Interpret the product on the conversion chain, retaining its actual scale Δ'.

Table 11. Operation counts at $N = 2^{16}$. In each tuple, the first entry counts key switches and the second counts plaintext–ciphertext multiplications in linear transformations. Galois keys are shared by the two procedures of each scheme.

$\log n$	Ours			GZ		
	LazyFlatten	ArithBoot	Galois keys	A2A-I	A2A	Galois keys
3	(90, 638)	(158, 803)	66	(118, 697)	(264, 1 394)	75
6	(50, 190)	(124, 370)	40	(152, 921)	(332, 1 842)	73
9	(30, 78)	(114, 298)	50	(272, 2 713)	(572, 5 426)	84

2. Apply lazy flattening with $(\Delta/\Delta') \cdot \mathbf{V}_{N\leftarrow n}$ over the two shared levels, raise to level 12, and output a ciphertext at level 11 with scale $\Delta = 2^{50}$. Coefficient exposure consumes eight levels and produces two ciphertexts at level 3.
3. For each ciphertext, the first multi-value functional bootstrapping call raises to level 29 and consumes 19 levels: three for CoeffToSlot, eight for the complex exponential, and eight for the bit LUTs over $\mathbb{Z}_{128}$ (recall that $\beta = 2$ in our implementation). These LUTs incorporate the complements needed by align_r, leaving only rotations. Five three-to-two compressors consume eight levels in four stages; SlotToCoeff consumes the remaining two.
4. The second functional bootstrapping call raises to level $17 + \log n$ and consumes 14 levels for its transformations and LUTs over $\mathbb{Z}_4$. FinalDigitComb finishes at level 2, avoiding rescaling by the shared primes, whose sizes differ from the conversion scale.

Each functional bootstrapping call applies SlotToCoeff, ModRaise under its ephemeral secret, and CoeffToSlot, then evaluates a Chebyshev approximation of the complex exponential on $[-16, 16]$ with two double-angle steps, followed by the LUT polynomials. The LUT for bit r depends only on its argument modulo 2^{r+1}; evaluating it in the corresponding power of the exponential reduces its degree from 192 to $3 \cdot 2^r$.

Range and Precision. Under the model of Supplementary Material B.1, a raised coefficient divided by q_0 has absolute value at most 16.5. Here the message occupies the full period, so we bound the probability of leaving the exponential approximation interval $[-16, 16]$ directly. For $N = 2^{16}$, a union bound over all coefficients and four bootstrapping calls gives $8N \cdot (1/2)^{33}/33! \approx 2^{-136.7}$.

The LUT polynomials interpolate the desired values with zero first derivatives. In a separate bit-extraction precision test at RLWE dimension $N = 4\,096$, we vary the input-error bound $\varepsilon_{\mathsf{in}} > 0$, measured in units of the spacing between consecutive LUT inputs. The base-two logarithm of the

Table 12. Arithmetic bootstrapping step times at $N = 2^{16}$ and $t = X - 2$. Totals include multiplication by $t(X^{N/n})$ and final subtraction. They differ from Table 4 by about 1% and from Table 14 by less than 1%.

Operation	AWS		MacBook Pro	
	$\log n = 3$	$\log n = 9$	$\log n = 3$	$\log n = 12$
$\mathbf{V}_{N\leftarrow n}$	0.63 s	0.17 s	0.30 s	0.03 s
ModRaise, key switching	0.23 s	0.23 s	0.13 s	0.12 s
$\mathbf{W}_{2\leftarrow N}$	3.32 s	3.25 s	1.81 s	1.64 s
EvalMod	2.53 s	2.49 s	1.41 s	1.26 s
$\mathbf{U}_{n\leftarrow 2}$	0.13 s	0.51 s	0.07 s	0.43 s
$\mathbf{W}_{n\leftarrow N}$	1.12 s	0.33 s	0.56 s	0.06 s
Total	7.96 s	6.98 s	4.29 s	3.55 s
Share of $\mathbf{W}_{2\leftarrow N}$ and EvalMod	73%	82%	75%	82%

largest output bit error is about $3 + 2 \log \varepsilon_{\mathsf{in}}$, with a floor near -35. Compression and final addition increase its logarithm by a few bits.

In addition to Section 8.4, we tested the full multiplication-to-digit pipeline on random messages at $n = 64$ in 32 MacBook Pro runs across four dimensions. Every digit matched the product modulo $2^{64} + 1$. Table 13 records the largest distance from an integer over all stages and runs; the maximum always occurs in the two vectors after compression.

Table 13. Arithmetic-to-digit conversion checks at $n = 64$ on the MacBook Pro. The last column gives the base-two logarithm of the largest distance from an integer over all stages and runs.

N	Runs	Digits compared	Log largest distance from an integer
4 096	20	81 920	-23.6
8 192	5	40 960	-22.1
2^{14}	5	81 920	-20.5
2^{15}	2	2^{16}	-20.5

B.5 Measurements on the MacBook Pro

These measurements use a 14-core Apple M4 Pro, 24 GiB of memory, macOS 15.7, and Go 1.26, with one thread and means of ten timed iterations after warm-up. Our implementation fits in memory for all tested degrees up to $n = 2^{14}$. On the MacBook Pro, the original GZ prototype triggers disk swapping at every tested lane degree, while the port approaches the memory limit, causing substantial timing variation. The main comparisons therefore use AWS; the measurements below provide additional context.

Refreshing and Memory. Table 14 separates MacBook Pro timings from the published x86/HEXL results. Against the Lattigo port on the MacBook Pro, lazy flattening is 4.2× to 15.9× faster for $8 \leqslant n \leqslant 128$, and arithmetic bootstrapping is 2.2× to 3.8× faster for $8 \leqslant n \leqslant 64$. The port fits A2A-$I$ only up to $n = 128$ and A2A only up to $n = 64$: A2A retains the encoded diagonals of both A2A-I and A2A-e. Hoisted baby-step giant-step evaluation stores these diagonals in the extended key-switching modulus. The original prototype also stores one plaintext per diagonal, with memory growing linearly in n (Table 15).

Table 14. Refreshing times at $N = 2^{16}$ and $t = X - 2$. Ours and the GZ port run on the MacBook Pro; the x86 columns give published OpenFHE timings with Intel HEXL [GZ26]. Dashes mark degrees beyond those in [GZ26] (HEXL) and port runs that do not fit in the 24 GiB memory.

$\log n$	Overflow refresh			Overflow & error refresh		
	LazyFlatten	A2A-I (port)	A2A-I (x86)	ArithBoot	A2A (port)	A2A (x86)
3	0.90 s	3.74 s	4.3 s	4.30 s	9.31 s	10.1 s
4	0.75 s	3.81 s	4.7 s	4.19 s	9.39 s	10.9 s
5	0.64 s	4.48 s	5.3 s	4.13 s	10.41 s	11.8 s
6	0.42 s	6.38 s	6.4 s	3.89 s	14.93 s	13.5 s
7	0.39 s	6.28 s	7.7 s	3.89 s	–	15.3 s
8	0.35 s	–	10.2 s	3.94 s	–	19.5 s
10	0.21 s	–	–	3.88 s	–	–
12	0.12 s	–	–	3.53 s	–	–
14	0.07 s	–	–	3.68 s	–	–

Without Intel HEXL on the MacBook Pro, the original prototype's A2A takes 60.4, 62.6, 64.7, 68.9, and 77.7 seconds at $n = 8$, 16, 32, 64, and 128. Its A2A-I takes 12.2 seconds at $n = 8$. At this degree, our port's times for both procedures are within 15% of the published x86/HEXL values.

Table 15. Peak memory footprint of the original GZ OpenFHE prototype on the MacBook Pro at $N = 2^{16}$. Each process includes key generation, precomputation, and calls to A2A-I, A2A-e, and A2A.

$\log n$	3	4	5	6	7	8
Peak memory (GiB)	29.9	30.7	32.2	35.5	42.2	56.6

Other Message Spaces and Sustained Arithmetic. Table 16 repeats the paired finite-field and integer measurements of Section 8.2; their refreshing times remain close. For sustained arithmetic, Table 17 uses the schedule and cost formulas of Section 8.3. Direct timings of our evaluation loop are 1.99, 1.76, and 1.95 seconds per multiplication at $n = 8$, 64, and 4 096, within 4% of these estimates.

Table 16. Refreshing times on the MacBook Pro at $N = 2^{16}$ for extension fields and the corresponding integer message spaces. Each of our lanes holds one element for $\mathbb{F}_{5^{16}}$ and $\mathbb{F}_{37^{32}}$, and sixteen for $\mathbb{F}_{257^8}$.

Field	$\log n$	Finite-field messages			Modular-integer baseline		
		t	LazyFlatten	ArithBoot	t	LazyFlatten	ArithBoot
$\mathbb{F}_{5^{16}}$	5	$X^{16} - 2$	0.64 s	4.12 s	$X - 2$	0.64 s	4.12 s
$\mathbb{F}_{37^{32}}$	6	$X^{32} - 6$	0.42 s	3.96 s	$X - 6$	0.44 s	3.99 s
$\mathbb{F}_{257^8}$	10	$X^{128} - 2$	0.21 s	3.87 s	$X - 2$	0.21 s	3.88 s

C LLM Usage Disclosure

The authors developed all research ideas, theoretical results, proofs, and the initial draft. Large language models (LLMs) served as auxiliary tools under their supervision. The examples below paraphrase representative requests rather than reproduce an exhaustive prompt log. The authors checked, edited, and verified all incorporated outputs and take full responsibility for the manuscript.

- *Manuscript Review.* Claude Opus 5 and 5.5 assisted with checks of notation, cross-references, and agreement between the manuscript and experimental logs. Examples include "Identify symbols used with different meanings and check the cross-references" and "Check that the numbers in the abstract and introduction match the experimental results."
- *Language and Structure.* ChatGPT (GPT-6 Astra) helped revise, shorten, and organize the manuscript, including the abstract and experimental section. Examples include "Make the abstract accessible to a cryptography researcher less familiar with CKKS" and "Shorten this section while preserving its technical content and notation." Gemini 3.1 Pro assisted with grammar and academic style, for example, "Improve the grammar and flow of this paragraph without changing its meaning."
- *Support for Implementation and Verification.* Claude Opus 5 and 5.5 helped compare implementation versions, generate verification scripts, tests, and benchmark tools on top of existing implementations, and organize the artifact. Examples include "Compare these implementations and summarize their differences" and "Using the supplied scripts and proofs, recompute the tables and error bounds and compare them with the manuscript."

Table 17. Cycle lengths k and estimated amortized multiplication times on the MacBook Pro at $N = 2^{16}$ and $t = X - 2$, using the formulas of Section 8.3. Each multiplication is followed by lazy flattening or A2A-I; each cycle ends with arithmetic bootstrapping or A2A-e. The GZ port does not fit in the MacBook Pro's 24 GiB memory at $\log n = 12$.

$\log n$	Ours			GZ			Gain
	k	Per ciphertext	Per message	k	Per ciphertext	Per message	
3	4	1.99 s	0.24 ms	2	6.62 s	0.81 ms	3.3×
6	3	1.74 s	1.70 ms	1	15.0 s	14.6 ms	8.6×
12	2	1.89 s	118 ms	–	–	–	–