

High-Precision Homomorphic ALU over Arbitrary Moduli with $O(1)$ Bootstrapping

Jiaming Liu, Shihe Ma, Anyu Wang, and Xiaoyun Wang

Tsinghua University, Beijing, China
{liujm25,msh24}@mails.tsinghua.edu.cn
{anyuwang,xiaoyunwang}@tsinghua.edu.cn

Abstract. Homomorphic computation on large integers requires both arithmetic and non-arithmetic (e.g., Boolean) operations. The radix-based method by Cha et al. (EUROCRYPT’26) supports both operation types over arbitrary n -bit integers with a complexity of $O(\log n)$ and $O(1)$ bootstrapping, respectively. Meanwhile, the triangle encoding method by Gao and Zheng (CRYPTO’26) has a complexity of $O(1)$ bootstrapping in arithmetic mode, but is restricted to power-of-two integers. Its arithmetic-to-Boolean (A2B) conversions also needs $O(n)$ bootstrapping and is only $O(1)$ in the amortized sense.

This work improves both methods by supporting computation over arbitrary moduli with $O(1)$ bootstrapping for both operation types, and is a generalization of Gao and Zheng’s approach.

Our first contribution is a framework for efficient homomorphic arithmetic over arbitrary plaintext moduli. It supports flexible radix representations and uses a CVP-based method to find defining polynomials with well-conditioned canonical embeddings and small norm growth under ring operations. When the method fails to find suitable defining polynomials, we use homomorphic Montgomery multiplication over radix-friendly plaintext rings, where each Montgomery reduction requires only $O(1)$ bootstrapping calls. Our second contribution is an efficient arithmetic-to-digit (A2D) conversion using $O(1)$ bootstrapping calls, independent of the radix used by the arithmetic representation.

We implemented our method in OpenFHE. For P-384 and Curve25519, our arithmetic multiplication achieves $8.67 \sim 10.26\times$ and $12.57 \sim 13.41\times$ lower latency than CPL26’s LazyMult and ExactMult, respectively. The speedups increase to $17.47 \sim 27.24\times$ and $25.35 \sim 35.63\times$ in the amortized setting. For RSA-1024 and RSA-2048, our arbitrary-modulus multiplication achieves $4.25\times$ and $7.58\times$ lower latency and $16.74\times$ and $29.85\times$ lower amortized time than CPL26’s ExactMult, respectively. For 64-, 128-, and 256-bit messages, our A2B conversion is $3.75\times$, $6.40\times$, and $14.17\times$ faster than GZ26, respectively.

1 Introduction

Fully homomorphic encryption (FHE) enables computation directly on encrypted data and provides a powerful foundation for privacy-preserving computation [22].

Over the past decade, several families of FHE schemes have been developed, each optimized for a particular computation model. For example, BGV and BFV provide exact SIMD arithmetic over finite plaintext rings [12,19], CKKS supports approximate SIMD arithmetic over complex numbers [15], while FHEW and TFHE are designed for Boolean computation with fast bootstrapping [18,16].

Despite this diversity, existing FHE schemes remain highly specialized and do not yet provide an efficient counterpart of the general-purpose computation readily available on plaintext data. Three limitations are particularly relevant to integer computation. First, high-precision computation is expensive, because increasing the precision typically leads to faster noise growth, larger parameters, and more costly bootstrapping [16,14]. Second, arithmetic and non-arithmetic operations are usually efficient in different ciphertext representations. Supporting both within the same computation therefore often requires costly conversions between arithmetic-friendly and Boolean-friendly ciphertexts [10,27,30,26]. Third, the choice of plaintext modulus is often constrained by the underlying scheme. For instance, BGV/BFV achieve their best SIMD packing efficiency only for a small class of plaintext primes, whereas FHEW/TFHE mostly operate with power-of-two plaintext moduli. Consequently, simultaneously supporting *high precision, mixed arithmetic and non-arithmetic computation, and arbitrary integer moduli* remains challenging.

A substantial amount of work has explored high-precision SIMD homomorphic computation. In the direction of BGV/BFV, GBFV [21], dBFV [29], and REFHE [11] extend BFV-style encoding to efficiently support high-precision integer computation with slow noise growth. In another direction, discrete CKKS stores integers within CKKS slots and serves as the foundation for high-precision integer arithmetic [17,4,5]. Building on this framework, several recent works have pushed the supported precision substantially further and extended the computation model.

Boneh and Kim decompose large integers into small residue numbers before encoding them into discrete CKKS ciphertexts, enabling homomorphic computation on integers with thousands of bits [7]. Kim takes a different approach of representing integers modulo B^k with their radix- B digits [23,24]. Because arithmetic causes the underlying digit representation to grow in both size and length, CKKS bootstrapping is used to reduce it back to a (relaxed) small-norm representation. The original construction requires $O(k \log k)$ bootstrappings per ciphertext multiplication, which is later reduced to $O(\log k)$. Cha et al. [13] further improved this radix-based approach by introducing a faster bootstrapping procedure that restores either a relaxed or exact small-norm representation using $O(\log k)$ bootstrappings. They also extend the construction to arbitrary plaintext moduli through homomorphic Montgomery reduction.

More recently, Gao and Zheng propose the triangle encoding, a combination of discrete-CKKS with the REFHE representation. They simulate arithmetic in $\mathbb{Z}[X]/(X^k - X + 2, X - 2)$ within CKKS slots, thereby realizing SIMD arithmetic over $\mathbb{Z}_{2^k}$ [20]. Their construction supports both arithmetic and Boolean operations:

Table 1. Comparison of high-precision homomorphic integer computation methods. n is the bit length and $k = \Theta(n)$ is the number of radix digits.

Scheme	Type	Arithmetic	Non-arithmetic	Arbitrary modulus
REFHE [11]	Scalar	$O(n)$	$O(n)$	No
Kim25 [23]	SIMD	$O(k \log k)$	$O(k)$	No
Kim26 [24]	SIMD	$O(\log k)$	$O(k)$	No
CPL26 [13]	SIMD	$O(\log k)$	$O(1)^\dagger$	Yes
GZ26 [20]	SIMD	$O(1)$	$O(k)$	No
Ours	SIMD	$O(1)$	$O(1)$	Yes

[†] Non-arithmetic operations only achieve $O(1)$ complexity for small-norm inputs, and cost $O(\log k)$ on ciphertexts after multiplication.

arithmetic-domain bootstrapping requires only $O(1)$ CKKS bootstrappings, while arithmetic-to-Boolean (A2B) conversion requires $O(k)$ bootstrappings.

Among these discrete-CKKS-based constructions, the radix-based method of Cha et al. and the triangle encoding method of Gao and Zheng represent two complementary state-of-the-art approaches to high-precision homomorphic integer computation. The former supports arbitrary plaintext moduli and needs only a constant number of bootstrappings for non-arithmetic operations, but requires $O(\log k)$ bootstrappings for arithmetic operations. The latter achieves $O(1)$ bootstrapping in arithmetic operations, but is restricted to power-of-two moduli and requires $O(k)$ bootstrappings for A2B conversion. Therefore, no existing construction simultaneously provides high-precision computation over arbitrary integer moduli and constant-complexity support for both arithmetic and non-arithmetic operations.

This raises the following question:

Can we support high-precision computation over arbitrary integer moduli, with $O(1)$ bootstrappings for both arithmetic and non-arithmetic operations?

We answer this question affirmatively. Table 1 compares our method with existing works.

1.1 Our Contributions

Our work makes the following contributions.

- **FHE for arbitrary integer moduli.** We introduce a polynomial representation for modular integer computation together with a CVP-based method for finding suitable defining polynomials. It supports leveled SIMD arithmetic with one CKKS level per multiplication, and admits arithmetic bootstrapping that refreshes approximation error and polynomial overflow separately or jointly. For moduli without a suitable defining polynomial, we utilize homomorphic Montgomery reduction for efficient computation. By expressing the

digit reduction as linear transformations of the arithmetic representation, we avoid the radix carry procedure required by CPL26 and obtain arbitrary modulus multiplication with only $O(1)$ bootstrap calls.

- **Faster ALU with $O(1)$ bootstrapping.** We give arithmetic-to-digit conversion using exactly two functional bootstrap calls and logarithmic multiplicative depth, together with a digit-to-arithmetic conversion requiring only linear operations. The arithmetic-to-Boolean conversion can be realized in the same number of bootstrap calls and multiplicative depth. For $M = 2^n$, the same construction directly outputs the bits of the integer, giving arithmetic-to-Boolean conversion with two functional bootstrap calls and enabling a SIMD ALU that combines arithmetic and Boolean computation.
- **Implementation and evaluation.** We implement our constructions in OpenFHE and evaluate them across different integer moduli and arithmetic bases. For P-384 and Curve25519, our arithmetic multiplication achieves up to $10.26\times/13.41\times$ lower latency and $27.24\times/35.63\times$ lower amortized cost than CPL26’s LazyMult/ExactMult. For RSA-1024 and RSA-2048, our arbitrary-modulus multiplication achieves $4.25\times/7.58\times$ lower latency and $16.74\times/29.85\times$ lower amortized cost than CPL26’s ExactMult. Our A2B conversion is $3.75\times$, $6.40\times$, and $14.17\times$ faster than GZ26 for 64-, 128-, and 256-bit messages, respectively.

1.2 Technical Overview

Arithmetic Polynomial Representation and Packing into CKKS Slots.

Let $M \geq 2$ be an integer modulus. We choose an integer $b \geq 2$ and a monic polynomial $F_M(X)$ of even degree d satisfying $F_M(b) = M$, and define $\mathcal{P}_M = \mathbb{Z}[X]/\langle F_M(X) \rangle$. For $t = X - b$, $\mathcal{P}_M/\langle t \rangle \simeq \mathbb{Z}_M$, and evaluation at $X = b$ is a ring homomorphism from $\mathcal{P}_M$ to $\mathbb{Z}_M$. We do not require M to be a power of b .

For $0 \leq m < M$, write $m = m_0 + \kappa b^d$, where $\kappa = \lfloor m/b^d \rfloor$ and $m_0 = \sum_{i=0}^{d-1} m_i b^i$ with $0 \leq m_i < b$. Intuitively, κ and m_0 are the quotient and remainder from the Euclidean division of m by b^d . Writing $F_M(X) = X^d + \sum_{i=0}^{d-1} f_i X^i$, we encode $m \in \mathbb{Z}_M$ into $\mathcal{P}_M$ as

$$[m]_{F_M}(X) = \sum_{i=0}^{d-1} m_i X^i - \kappa (F_M(X) - X^d).$$

Its coefficients are $[m]_{F_M, i} = m_i - \kappa f_i$, and equivalently, the radix- b digits are $m_i = [m]_{F_M, i} + \kappa f_i$. Moreover, $[m]_{F_M}(b) = m - \kappa M$, and when $M \leq b^d$, we have $\kappa = 0$, so the coefficients are exactly the radix- b digits of m . Any polynomial representing the same residue m can be written as an *arithmetic polynomial* $A = [m]_{F_M} + tI$ for some *overflow polynomial* $I \in \mathcal{P}_M$. The value of I does not affect the encoded $\mathbb{Z}_M$ value, but the size of I will grow during homomorphic addition and multiplication.

We require F_M to have no repeated roots, so it has $d/2$ pairs of conjugate roots. Evaluation at one root from each pair embeds $\mathcal{P}_M$ into $\mathbb{C}^{d/2}$, and these values are stored directly in CKKS slots, allowing one ciphertext to pack $\lfloor N/d \rfloor$ elements

from $\mathcal{P}_M$. Because the embedding preserves ring operations, CKKS arithmetic evaluates the corresponding modular arithmetic SIMD-wise. For efficient homomorphic evaluation, we prefer defining polynomials F_M with well-conditioned embeddings and small norm growth during ring operations. Since the polynomials satisfying $F_M(b) = M$ form a lattice in the coefficient space, finding a suitable F_M can be reduced to a CVP over a suitably constructed lattice.

If no suitable defining polynomial is available for a modulus M , we instead use fixed arithmetic spaces together with Montgomery representation. The reduction is expressed through linear transformations of the arithmetic representation, and the bounded ambiguity in the Montgomery quotient is absorbed into the redundant representation. Division by b^n is incorporated into the final arithmetic bootstrap. Thus, multiplication in $\mathbb{Z}_M$ can be efficiently implemented using one ciphertext multiplication and one arithmetic bootstrap.

Bootstrapping arithmetic ciphertexts. Homomorphic arithmetic increases both the overflow polynomial and the CKKS approximation error, which can be refreshed separately or together. The overflow refresh does not eliminate the overflow polynomial, but replaces it with a new bounded one. We multiply by $(X - b)^{-1}$ to separate the integer overflow from the message, reduce the resulting integer part modulo a small modulus, and use modulus raising to return to the arithmetic representation with bounded overflow. The approximation error can be refreshed separately by isolating the error, bootstrapping it, and subtracting the result from the original ciphertext, leaving the same arithmetic polynomial with a smaller error. When both refreshes are required, we fuse the two procedures so that they share the linear transformations.

Conversion between Arithmetic and Radix Digits. In the arithmetic representation, the CKKS slots contain the root evaluations of the polynomial $A = [m]_{F_M} + (X - b)I$. For a radix $B = b^s$, the arithmetic-to-digit conversion puts the radix- B digits of m into CKKS slots. When $M \leq b^d$, the representative is straightforward and the coefficients of $[m]_{F_M}$ are exactly the radix- b digits of m . For a general modulus, the representative contains an additional bounded term, which is handled by the same conversion.

The reverse digit-to-arithmetic conversion requires only a linear transformation. Given radix- B digits d_j , we form $\sum_j d_j X^{sj} \bmod F_M(X)$. Since its evaluation at $X = b$ is congruent to $\sum_j d_j b^{sj} = \sum_j d_j B^j = m$ modulo M , this polynomial represents the same integer in $\mathcal{P}_M$. Since each $X^{sj} \bmod F_M(X)$ is public, the corresponding root evaluations are obtained linearly from the radix digits.

We now describe the main idea of arithmetic-to-digit conversion for $M = 2^n$ and $B = 2^s$, specialized for Boolean output. Assume that $s \mid n$ and let $\ell = n/s$. We take $F_M(X) = X^n - X + 2$ and $t = X - 2$, and write $m = \sum_{i=0}^{n-1} m_i 2^i$. After refreshing the overflow polynomial, let $A' = [m]_{F_M} + tI' = \sum_{i=0}^{n-1} A_i X^i$, where I' is bounded. A linear transformation then combines every s consecutive coefficients into a radix- 2^s digit. Let $B = 2^s$ and $d_j \in \{0, \dots, B - 1\}$ be the j -th

radix- B digit of m , also set $x_j = \sum_{r=0}^{s-1} 2^r A_{sj+r}$. The intermediate overflow terms cancel under these weights, giving

$$x_j = d_j + I'_{sj-1} - BI'_{s(j+1)-1},$$

where $I'_{-1} = 0$. Thus, each x_j contains the desired radix digit and two coefficients of the bounded overflow polynomial.

We use a functional bootstrap to refresh the overflow and compute $x_j \bmod B$. Let r_j be the resulting remainder and write $x_j = r_j + Bq_j$, where $q_j = (x_j - r_j)/B$ is obtained from a simple linear operation. For an appropriate choice of remainder representatives, there is a carry c_{j+1} such that

$$r_j = d_j + I'_{sj-1} - Bc_{j+1}, \quad q_j = c_{j+1} - I'_{s(j+1)-1}.$$

Let $\mathcal{C}$ denote the public finite set of all possible carry values determined by the remainder representatives and the bound on I' . Setting $c_0 = 0$, $a_0 = r_0$, and $a_j = r_j + q_{j-1}$ for $1 \leq j < \ell$ cancels the common I'_{sj-1} term and gives

$$a_j = d_j + c_j - Bc_{j+1}.$$

This recurrence relation is free of the unknown overflow polynomial I' , thereby reducing the recovery of the digits d_j to determining the carries c_j .

For simplicity, consider the binary-carry case $\mathcal{C} = \{0, 1\}$. For a possible incoming carry $c \in \mathcal{C}$, define $f_j(c) = \lceil (c - a_j)/B \rceil$. The relation transforms into

$$f_j(c_j) = c_{j+1}, \quad a_j - c_j + Bf_j(c_j) = d_j.$$

We then use a multi-value functional bootstrap over encryption of a_j to produce multiple ciphertexts. Inside the slots, two of them contain $f_j(0)$ and $f_j(1)$ for every j , respectively. For each $0 \leq k < s$, the remaining $2s$ ciphertexts contain

$$b_{j,k}(c) = \text{bit}_k(a_j - c + Bf_j(c)), \quad c = 0, 1.$$

where $\text{bit}_k(x)$ denotes the k -th binary bit of x , with $k = 0$ being the least significant bit. When $c = c_j$, $b_{j,k}(c_j)$ gives the desired value of $\text{bit}_k(d_j) = m_{sj+k}$. Therefore, for each j , these ciphertexts contain all candidates of $f_j(c)$ and $b_{j,k}(c)$ over all possible choices of $c \in \{0, 1\}$.

It remains to determine which candidate corresponds to the actual carry. Suppose that the incoming carry at position j is c and let $y = f_j(c)$. Then y is the carry entering position $j + 1$, so the next carry and the bits at position $j + 1$ are selected as

$$f_{j+1}(y) = (1-y)f_{j+1}(0) + yf_{j+1}(1), \quad b_{j+1,k}(y) = (1-y)b_{j+1,k}(0) + yb_{j+1,k}(1).$$

Hence, two consecutive radix positions can be combined so that both the outgoing carry and the corresponding output bits depend solely on the carry entering the first position.

This composition can be continued in a binary-tree manner. At each multiplicative level, the length of each composed range doubles. After $\lceil \log_2 \ell \rceil$ levels,

the carry and bit outputs at every radix position depend only on the initial carry c_0 . Since $c_0 = 0$ is known, evaluating the composed outputs at $c_0 = 0$ gives m_{sj+k} for every j and $0 \leq k < s$. A final linear transformation rearranges these bits into the Boolean representation.

Thus, arithmetic-to-Boolean conversion uses exactly two functional bootstrap calls followed by $O(\log \ell)$ multiplicative depth.

1.3 Organization

Section 2 reviews the FHE and CKKS tools used in our construction. Section 3 introduces the arithmetic polynomial representation, CKKS packing, homomorphic arithmetic, and the defining-polynomial search procedure. Section 4 presents overflow refresh, error refresh, complete arithmetic bootstrapping, and multiplication over arbitrary moduli. Section 5 introduces the digit representation and conversions between arithmetic and digit representations, including arithmetic-to-Boolean conversion as a special case and its extension to general integer moduli. Section 6 reports our implementation and experimental evaluation. Appendix A gives the detailed proofs, Appendix B gives the defining-polynomial search and example constructions, and Appendix C gives the detailed conversion algorithm and analysis.

2 Preliminaries

2.1 Notation

For an integer $q \geq 2$, let $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. Let N be a power of two and define $\mathcal{R} = \mathbb{Z}[X]/\langle X^N + 1 \rangle$. For an integer modulus $Q \geq 2$, let $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$ and $\mathcal{R}_{\mathbb{R}} = \mathcal{R} \otimes_{\mathbb{Z}} \mathbb{R}$. For a polynomial or vector a , $\|a\|_{\infty}$ denotes its coefficient or component ℓ_{∞} -norm. Matrices are denoted by uppercase bold letters and vectors by lowercase bold letters. All vectors are column vectors. The identity matrix of dimension k is denoted by $\mathbf{I}_k$. The symbol $\odot$ denotes componentwise multiplication. For a statement P , let $\mathbf{1}[P]$ be 1 if P holds and 0 otherwise. All logarithms are to base two. For a nonnegative integer a , let $\text{bit}_k(a) = \lfloor a/2^k \rfloor \bmod 2$ denote its k -th bit, indexed from $k = 0$.

2.2 CKKS Scheme

Encoding and encryption. CKKS [15] is an approximate homomorphic encryption scheme based on RLWE over the ring $\mathcal{R}$. Let $\zeta = \exp(\pi i/N)$ be a primitive $2N$ -th root of unity and let $\zeta_j = \zeta^{5^j}$ for $0 \leq j < N/2$. For $m(X) \in \mathcal{R}_{\mathbb{R}}$, we have the canonical embedding

$$\sigma : \mathcal{R}_{\mathbb{R}} \longrightarrow \mathbb{C}^{N/2}, \quad \sigma(m) = (m(\zeta_0), \dots, m(\zeta_{N/2-1}))^{\top}.$$

The map σ is an isomorphism of real algebras, so addition and multiplication in $\mathcal{R}_{\mathbb{R}}$ correspond to componentwise operations in $\mathbb{C}^{N/2}$. Each component of

$\sigma(m)$ is called a CKKS slot. For $r \in \mathbb{Z}$, let τ_r denote the ring automorphism $\tau_r(m(X)) = m(X^{5^r})$, where the exponent is taken modulo $2N$. Under the canonical embedding, τ_r maps $(m(\zeta_j))_{0 \leq j < N/2}$ to $(m(\zeta_{(j+r) \bmod (N/2)}))_{0 \leq j < N/2}$. Thus, τ_r corresponds to a cyclic rotation of the CKKS slots by r positions.

For a scaling factor Δ , CKKS encodes a vector $\mathbf{z} \in \mathbb{C}^{N/2}$ as

$$\text{Ecd}(\mathbf{z}, \Delta) = \lfloor \Delta \sigma^{-1}(\mathbf{z}) \rfloor \in \mathcal{R}, \quad \text{Dcd}(m, \Delta) = \frac{\sigma(m)}{\Delta} \approx \mathbf{z}.$$

For a ciphertext modulus Q , let $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$. Let χ_s and χ_e denote the secret key and error distributions over $\mathcal{R}$, and sample $s \leftarrow \chi_s$. To encrypt a plaintext polynomial $m \in \mathcal{R}_Q$, sample $a \leftarrow \mathcal{R}_Q$ uniformly and $e \leftarrow \chi_e$, and set $b = -as + m + e \pmod{Q}$. Then

$$\text{ct} = \text{Enc}_s(m) = (b, a) \in \mathcal{R}_Q^2, \quad \text{Dec}_s(\text{ct}) = b + as = m + e \pmod{Q}.$$

In CKKS, $\text{Ecd}(\mathbf{z}, \Delta)$ is reduced modulo Q and encrypted in this form, and decoding after decryption yields an approximation of $\mathbf{z}$. For a slot vector $\mathbf{z} \in \mathbb{C}^{N/2}$, we write $\text{Enc}_s(\mathbf{z})$ as shorthand for $\text{Enc}_s(\text{Ecd}(\mathbf{z}, \Delta))$, and say that the resulting ciphertext encrypts $\mathbf{z}$.

Homomorphic operations and levels. CKKS uses a chain of ciphertext moduli to support leveled computation. Let $Q_\ell = q_0 q_1 \cdots q_\ell$ denote the ciphertext modulus at level ℓ , where $q_1, \dots, q_\ell$ are typically chosen with bit sizes close to $\log_2 \Delta$. For level analysis, we therefore view $Q_\ell \approx q_0 \Delta^\ell$.

We use the standard CKKS operations **Add**, **PMult**, **CMult**, **Rescale**, **Rot**, and **LinTrans** [15, 8]. **Add** and **PMult** perform ciphertext addition and ciphertext-plaintext componentwise multiplication, respectively. **CMult** performs ciphertext-ciphertext componentwise multiplication, and **Rescale** $_{Q_\ell \rightarrow Q_{\ell-1}}$ reduces the ciphertext modulus and scale; a ciphertext multiplication followed by rescaling consumes one CKKS level. **Rot** $_r$ cyclically rotates the encrypted slots by r positions using an automorphism followed by key switching. For a public matrix $\mathbf{A}$, **LinTrans** $_{\mathbf{A}}$ evaluates the linear map $\mathbf{m} \mapsto \mathbf{A}\mathbf{m}$ using rotations and plaintext multiplications.

SlotsToCoeffs and CoeffsToSlots. We use the linear transformations underlying CKKS bootstrapping to convert between the slot and coefficient representations [8]. For $m(X) = \sum_{i=0}^{N-1} m_i X^i \in \mathcal{R}_{\mathbb{R}}$, let $\mathbf{m} = (m_0, \dots, m_{N-1})^\top$ and define

$$\mathbf{U}_\sigma = (\zeta_j^i)_{\substack{0 \leq j < N/2, \\ 0 \leq i < N}}, \quad \begin{pmatrix} \mathbf{U}_\sigma \\ \overline{\mathbf{U}_\sigma} \end{pmatrix}^{-1} = [\mathbf{V}_\sigma \quad \overline{\mathbf{V}_\sigma}].$$

Then $\sigma(m) = \mathbf{U}_\sigma \mathbf{m}$. Write $\mathbf{m} = (\mathbf{m}^{(0)\top}, \mathbf{m}^{(1)\top})^\top$, where $\mathbf{m}^{(0)} = (m_0, \dots, m_{N/2-1})^\top$ and $\mathbf{m}^{(1)} = (m_{N/2}, \dots, m_{N-1})^\top$.

Let $\mathbf{V}_\sigma = (\mathbf{V}_{\sigma,0}^\top, \mathbf{V}_{\sigma,1}^\top)^\top$. The **SlotsToCoeffs** transformation **StC** $_{\mathcal{R}}$ maps a ciphertext encrypting $\sigma(m)$ to two ciphertexts encrypting

$$\mathbf{m}^{(h)} = 2 \text{Re}(\mathbf{V}_{\sigma,h} \sigma(m)), \quad h \in \{0, 1\}.$$

Conversely, let $\mathbf{U}_\sigma = [\mathbf{U}_{\sigma,0} \mid \mathbf{U}_{\sigma,1}]$. The `CoeffsToSlots` transformation $\text{CtS}_\mathcal{R}$ maps ciphertexts encrypting $\mathbf{m}^{(0)}$ and $\mathbf{m}^{(1)}$ to a ciphertext encrypting

$$\sigma(m) = \mathbf{U}_{\sigma,0}\mathbf{m}^{(0)} + \mathbf{U}_{\sigma,1}\mathbf{m}^{(1)}.$$

Both transformations are evaluated using `LinTrans`, while $\text{StC}_\mathcal{R}$ additionally uses homomorphic conjugation to extract the real parts.

2.3 Discrete CKKS and Functional Bootstrapping

Discrete CKKS. Discrete CKKS restricts the plaintext values in CKKS slots to a finite domain, such as bits or small integers [17,4,5]. For a plaintext modulus $b \geq 2$, we identify $m \in \mathbb{Z}_b$ with an integer in $\{0, \dots, b-1\}$. At scale Δ , the value m is represented in a CKKS slot by $\Delta m/b$. For a vector $\mathbf{m} \in \mathbb{Z}_b^{N/2}$, this representation is applied componentwise. A ciphertext encrypting $\mathbf{m}$ therefore decrypts in the slots to $\Delta \mathbf{m}/b + \mathbf{e}$, where $\mathbf{e}$ denotes the approximation error.

Functional bootstrapping. Functional bootstrapping refreshes a discrete CKKS ciphertext while evaluating a function on its encrypted slots [1]. Let $f : \mathbb{Z}_b \rightarrow \mathbb{C}$, and let ct encrypt $\mathbf{m} \in \mathbb{Z}_b^{N/2}$ at the bottom modulus q_0 .

The procedure first applies $\text{StC}_\mathcal{R}$ to move the encrypted slot values to polynomial coefficients. Each resulting ciphertext is then lifted from modulus q_0 to a larger modulus Q by $\text{ModRaise}_{q_0 \rightarrow Q}$. If an input to modulus raising decrypts to $m(X) + e(X)$ modulo q_0 , the raised ciphertext decrypts to

$$m(X) + e(X) + q_0 J(X) \pmod{Q},$$

for some integer polynomial $J(X) \in \mathcal{R}$ [9,20]. Applying $\text{CtS}_\mathcal{R}$ then moves these coefficients back to the slots, so each encrypted slot contains its original encoded value together with an integer multiple of q_0 .

Lemma 1 (Overflow after modulus raising). *If ModRaise uses an encapsulated sparse ternary secret key of Hamming weight 32, then $\|J\|_\infty \leq 15$ except with negligible probability [20].*

After modulus raising, $\text{CtS}_\mathcal{R}$ maps the coefficient ciphertexts back to the slot representation. For $f : \mathbb{Z}_b \rightarrow \mathbb{C}$, EvalLUT_f evaluates a q_0 -periodic trigonometric interpolation that takes the value $f(m)$ at $q_0 m/b$ for every $m \in \mathbb{Z}_b$. Since the interpolation is expressed in terms of $e^{2\pi i j x / q_0}$, it is invariant under shifts by multiples of q_0 , and thus evaluates $f(\mathbf{m})$ independently of the integer offsets introduced by modulus raising. We denote the complete procedure by

$$\text{FBS}_f = \text{EvalLUT}_f \circ \text{CtS}_\mathcal{R} \circ \text{ModRaise}_{q_0 \rightarrow Q} \circ \text{StC}_\mathcal{R},$$

where $\text{ModRaise}_{q_0 \rightarrow Q}$ is applied to each ciphertext output by $\text{StC}_\mathcal{R}$.

Multi-value functional bootstrapping. For several functions $f_1, \dots, f_t : \mathbb{Z}_b \rightarrow \mathbb{C}$ evaluated on the same ciphertext, multi-value functional bootstrapping shares the computation among all outputs [1]. The transformations $\text{StC}_{\mathcal{R}}$, ModRaise , and $\text{CtS}_{\mathcal{R}}$ are performed only once, and the Fourier terms required by the trigonometric interpolation are also computed once for the common encrypted input. Each function f_i has its own interpolation coefficients, which are then combined with these shared Fourier terms to produce a ciphertext encrypting $f_i(\mathbf{m})$. The shared computation dominates the cost of functional bootstrapping. Thus, a single functional bootstrapping procedure can produce multiple function outputs while amortizing the dominant lookup evaluation across them.

2.4 Triangle Encoding

In [20], they introduce the triangle encoding for arithmetic over $\mathbb{Z}_{2^n}$. They use the ring $\mathcal{P}_n = \mathbb{Z}[X]/\langle X^n - X + 2 \rangle$ to represent arithmetic messages and perform arithmetic operations. We briefly review their encoding and the operations.

Encoding. For $m = \sum_{i=0}^{n-1} m_i 2^i \in \mathbb{Z}_{2^n}$, define $[m]_t = \sum_{i=0}^{n-1} m_i X^i$. They encode m in $\mathcal{P}_{n,\mathbb{R}}$ as $\Delta t^{-1} [m]_t$ and they call the polynomial $\Delta t^{-1} [m]_t$ the triangle encoding of m . The general form of the triangle encoding is

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

where I is the arithmetic overflow polynomial and e is the approximation error. They pack N/n polynomials in $\mathcal{P}_n$ in a single CKKS ciphertext. For Boolean encoding, they place the bits $m_0, \dots, m_{n-1}$ directly in CKKS slots. Each slot therefore encrypts one message bit, and the bits of multiple messages are packed into the same ciphertext. Boolean operations are then evaluated slotwise on these encrypted bits.

Arithmetic operations. Addition is performed directly and preserves the triangle form. For two plaintexts f and g encoding m and m' , respectively, multiplication is defined as

$$\frac{1}{\Delta^2} \Delta t f g = \frac{t}{\Delta} f g.$$

The factor t/Δ preserves the triangle form, and the result encodes $mm' \bmod 2^n$ with a new overflow polynomial and error. The operation multiplies f and g , multiplies the product by Δt , and rescales twice, so a full multiplication in triangle encoding consumes two CKKS levels.

When some arithmetic operations are performed on a ciphertext, the overflow polynomial I and the error e will grow. Thus, they support arithmetic refresh to reduce the overflow polynomial and approximation error while preserving the message.

Arithmetic-to-Boolean Conversion. The conversion maps the triangle encoding to CKKS slots and extracts the message bits in groups of w bits using discrete CKKS functional bootstrapping [20]. Each iteration extracts one group and removes its contribution before processing the next group. Thus, converting a single n -bit ciphertext requires $O(n/w)$ CKKS bootstrapping operations. The batched version processes the same group from several ciphertexts in one bootstrapping operation, reducing the amortized number of bootstrapping operations to $O(1)$ per ciphertext.

3 Arithmetic Polynomial Encoding

3.1 Polynomial Representation

Let $M \geq 2$ be an integer modulus, let $b \geq 2$, and let $F_M(X) \in \mathbb{Z}[X]$ be a monic polynomial of even degree d satisfying $F_M(b) = M$ and having d distinct nonreal roots. Define $\mathcal{P}_M = \mathbb{Z}[X]/\langle F_M(X) \rangle$ and $t = X - b$. Since $F_M(b) = M$, evaluation at $X = b$ gives

$$\mathcal{P}_M/\langle t \rangle \simeq \mathbb{Z}[X]/\langle F_M(X), X - b \rangle \simeq \mathbb{Z}_M.$$

Thus, polynomial arithmetic in $\mathcal{P}_M$ represents arithmetic modulo M after evaluation at $X = b$. For $m \in \mathbb{Z}_M$, we identify m with its canonical representative $0 \leq m < M$ and write $m = m_0 + \kappa b^d$, where $0 \leq m_0 < b^d$ and $\kappa = \lfloor m/b^d \rfloor$. Writing the radix- b expansion $m_0 = \sum_{i=0}^{d-1} m_i b^i$ with $m_i \in \{0, \dots, b-1\}$, we define

$$[m]_{F_M}(X) = \sum_{i=0}^{d-1} m_i X^i - \kappa (F_M(X) - X^d). \quad (1)$$

Indeed, $[m]_{F_M}(b) = m - \kappa M \equiv m \pmod{M}$. When $M \leq b^d$, we have $\kappa = 0$ for every $m \in [0, M)$, so $[m]_{F_M}(X) = \sum_{i=0}^{d-1} m_i X^i$ and its coefficients are exactly the radix- b digits of m . When $M > b^d$, the bounded value κ folds the part above the lowest d radix- b positions into the degree- d polynomial.

For a fixed modulus M , representations under any two choices (b, F_M) and (b', F'_M) can be converted by a public linear transformation that preserves the represented value modulo M . Two polynomials in $\mathcal{P}_M$ represent the same residue modulo M exactly when their difference is a multiple of t . Hence, every polynomial representing m can be written as $A = [m]_{F_M} + tI$ for some $I \in \mathcal{P}_M$. We call I the *overflow polynomial*.

The overflow polynomial has a direct interpretation in terms of radix- b carries. For $i < d-1$, a carry c from position i to $i+1$ changes the polynomial by $cX^{i+1} - bcX^i = cX^i(X-b)$. A carry from position $d-1$ increases κ and changes the polynomial by $-c(F_M(X) - X^d) - bcX^{d-1} \equiv cX^{d-1}(X-b) \pmod{F_M}$. Thus, tI records the changes introduced by radix- b carries, including the carry into κ .

We next describe how arithmetic polynomials in $\mathcal{P}_M$ are packed into CKKS slots.

3.2 Packing in CKKS Slots

We follow the same idea as in [20] to pack multiple encoded polynomials in $\mathcal{P}_M$ into one CKKS ciphertext. The key idea of the batching method is to simulate the ring $\mathcal{P}_M$ by $\mathbb{C}^{d/2}$, and since one CKKS plaintext has $N/2$ complex slots, we can pack $\lfloor N/d \rfloor$ polynomials in $\mathcal{P}_M$ in one plaintext.

Let $\xi_0, \dots, \xi_{d/2-1}$ be the roots of F_M in the upper half-plane, in a fixed order, and define

$$\rho : \mathcal{P}_{M,\mathbb{R}} \rightarrow \mathbb{C}^{d/2}, \quad \rho(A) = (A(\xi_0), \dots, A(\xi_{d/2-1}))^\top.$$

Since F_M has d distinct nonreal roots, ρ is an isomorphism of real algebras.

For $0 \leq r < \lfloor N/d \rfloor$, let $A_r = [m_r]_{F_M} + tI_r$ represent the r -th message. We concatenate their images under ρ into the CKKS slots:

$$\mathbf{z} = (\rho(A_0)^\top, \dots, \rho(A_{\lfloor N/d \rfloor - 1})^\top, 0, \dots, 0)^\top \in \mathbb{C}^{N/2},$$

and encode $\mathbf{z}$ at scale Δ as $\lfloor \Delta \sigma^{-1}(\mathbf{z}) \rfloor \bmod Q$.

Since ρ is an algebra isomorphism, CKKS additions and multiplications apply the corresponding operations independently to all packed arithmetic polynomials. After decryption, applying ρ^{-1} to the slots of message r gives $\Delta A_r + e_r$, where $e_r \in \mathcal{P}_{M,\mathbb{R}}$ is the CKKS approximation error.

Arithmetic ciphertexts store the evaluations $\rho(A)$ in CKKS slots. Some of our procedures instead operate directly on the coefficients of A . We therefore define $\text{EvalToCoeff}_{\mathcal{P}_M}$ to transform the slot representation $\rho(A)$ into CKKS slots containing the coefficients of A , and $\text{CoeffToEval}_{\mathcal{P}_M}$ for the inverse transformation.

Let $A(X) = \sum_{i=0}^{d-1} c_i X^i$ with coefficient vector $\mathbf{c} = (c_0, \dots, c_{d-1})^\top$. Define

$$\mathbf{U}_\rho = (\xi_j^i)_{\substack{0 \leq j < d/2, \\ 0 \leq i < d}}, \quad \begin{pmatrix} \mathbf{U}_\rho \\ \overline{\mathbf{U}_\rho} \end{pmatrix}^{-1} = [\mathbf{V}_\rho \quad \overline{\mathbf{V}_\rho}].$$

Then $\rho(A) = \mathbf{U}_\rho \mathbf{c}$ and $\mathbf{c} = 2 \text{Re}(\mathbf{V}_\rho \rho(A))$. Since A has d coefficients while $\rho(A)$ occupies $d/2$ CKKS slots, $\text{EvalToCoeff}_{\mathcal{P}_M}$ outputs two ciphertexts containing the first and second halves of the coefficients.

Partition $\mathbf{U}_\rho = [\mathbf{U}_{\rho,0} \mid \mathbf{U}_{\rho,1}]$ and $\mathbf{V}_\rho = (\mathbf{V}_{\rho,0}^\top, \mathbf{V}_{\rho,1}^\top)^\top$ into $(d/2) \times (d/2)$ blocks, and let $\mathbf{c}^{(0)} = (c_0, \dots, c_{d/2-1})^\top$ and $\mathbf{c}^{(1)} = (c_{d/2}, \dots, c_{d-1})^\top$. Then $\text{EvalToCoeff}_{\mathcal{P}_M}$ computes $\mathbf{c}^{(h)} = 2 \text{Re}(\mathbf{V}_{\rho,h} \rho(A))$ for $h \in \{0, 1\}$, placing these coefficient values in CKKS slots. Conversely, $\text{CoeffToEval}_{\mathcal{P}_M}$ reconstructs the arithmetic slot representation as $\rho(A) = \mathbf{U}_{\rho,0} \mathbf{c}^{(0)} + \mathbf{U}_{\rho,1} \mathbf{c}^{(1)}$.

For all packed messages, the same transformations are applied independently to each block of $d/2$ slots. They are evaluated homomorphically using LinTrans and Conj as in Section 2.3, and therefore use the same set of rotations for all packed messages.

Theorem 1 (Encoding error). *Let $\mathbf{z} \in \mathbb{C}^{N/2}$ be the packed slot vector, and let $\tilde{\mathbf{a}} = \lfloor \Delta \sigma^{-1}(\mathbf{z}) \rfloor$ be its CKKS encoding at scale Δ . For each packed message r , CKKS encoding gives $\Delta A_r + e_{\text{enc},r}$ after applying ρ^{-1} to its assigned slots, where $\|e_{\text{enc},r}\|_\infty \leq \|\mathbf{V}_\rho\|_\infty \|\sigma\|_\infty$.*

The proof, together with those of the arithmetic-growth theorems below, is given in Appendix A.1.

Rotations. Each packed arithmetic polynomial occupies $d/2$ consecutive CKKS slots. Therefore, rotating the packed messages by k positions corresponds to a slot rotation by $kd/2$, except that CKKS rotations wrap around all $N/2$ slots rather than only the $\lfloor N/d \rfloor$ occupied message blocks. When $d \mid N$, all slots are occupied, and $\text{Rot}_{kd/2}$ directly gives the desired cyclic rotation.

Otherwise, let $\mathbf{m}_k^{(0)}$ select the first $\lfloor N/d \rfloor - k$ message blocks and $\mathbf{m}_k^{(1)}$ select the remaining k blocks. The cyclic rotation is then

$$\mathbf{m}_k^{(0)} \odot \text{Rot}_{kd/2}(\text{ct}) + \mathbf{m}_k^{(1)} \odot \text{Rot}_{(k - \lfloor N/d \rfloor)d/2}(\text{ct}).$$

The first term performs the non-wrapping part of the rotation, while the second wraps the remaining messages within the occupied slot region.

Under this packing, CKKS addition and multiplication evaluate the corresponding operations on all packed arithmetic polynomials. We next describe the resulting homomorphic arithmetic and analyze the growth of the overflow polynomial and approximation error.

3.3 Homomorphic Arithmetic

We now consider addition and multiplication of arithmetic polynomial encodings. We write $\mathcal{P}_{M,\mathbb{R}} = \mathcal{P}_M \otimes_{\mathbb{Z}} \mathbb{R}$ and represent its elements by polynomials of degree less than d . For $m \in \mathbb{Z}_M$, let $[m]_{F_M}$ denote the polynomial representative defined above.

Let $\Delta A + e$ and $\Delta A' + e'$ be two approximate representations encoding m and m' , where $A = [m]_{F_M} + tI$ and $A' = [m']_{F_M} + tI'$. Since $F_M(b) = M$, evaluation at b induces a ring homomorphism from $\mathcal{P}_M$ to $\mathbb{Z}_M$ whose kernel is generated by t . Hence, $[m]_{F_M} + [m']_{F_M} = [m + m']_{F_M} + tC_+$ and $[m]_{F_M}[m']_{F_M} = [mm']_{F_M} + tC_{\times}$ for some $C_+, C_{\times} \in \mathcal{P}_M$.

Addition. The sum of $\Delta A + e$ and $\Delta A' + e'$ is $\Delta(A + A') + e + e'$. Since $[m]_{F_M} + [m']_{F_M} = [m + m']_{F_M} + tC_+$, we have $A + A' = [m + m']_{F_M} + t(I + I' + C_+)$. Thus, the sum encodes $m + m'$ at scale Δ with overflow polynomial $I + I' + C_+$ and approximation error $e + e'$.

Multiplication. The product of $\Delta A + e$ and $\Delta A' + e'$ is $\Delta^2 AA' + \Delta(Ae' + A'e) + ee'$. Since $[m]_{F_M}[m']_{F_M} = [mm']_{F_M} + tC_{\times}$, we have

$$AA' = [mm']_{F_M} + tI_{\times}, \quad I_{\times} = C_{\times} + I[m']_{F_M} + I'[m]_{F_M} + tII'.$$

After rescaling by Δ , the output encodes mm' at scale Δ with overflow polynomial $I_{\times}$ and approximation error $e_{\times} = Ae' + A'e + ee'/\Delta$.

Addition and multiplication increase the overflow polynomial and the approximation error. We next bound their growth.

Let μ_M be a constant such that $\|ab\|_{\infty} \leq \mu_M \|a\|_{\infty} \|b\|_{\infty}$ for all $a, b \in \mathcal{P}_{M,\mathbb{R}}$. Since $\|t\|_{\infty} = b$, this also gives $\|ta\|_{\infty} \leq b\mu_M \|a\|_{\infty}$.

Theorem 2 (Overflow and error growth under addition). *Let $\Delta A + e$ and $\Delta A' + e'$ be two approximate representations encoding m and m' . Their sum encodes $m + m'$ at scale Δ with overflow polynomial $I + I' + C_+$ and approximation error $e + e'$. Moreover,*

$$\|I + I' + C_+\|_\infty \leq \|I\|_\infty + \|I'\|_\infty + \|C_+\|_\infty, \quad \|e + e'\|_\infty \leq \|e\|_\infty + \|e'\|_\infty.$$

Theorem 3 (Overflow and error growth under multiplication). *Let $\Delta A + e$ and $\Delta A' + e'$ be two approximate representations encoding m and m' . After multiplication and division by Δ , the output is $\Delta A A' + e_\times$, where $e_\times = A e' + A' e + \frac{e e'}{\Delta}$. It encodes $m m'$ at scale Δ with overflow polynomial $I_\times$ satisfying*

$$\begin{aligned} \|I_\times\|_\infty &\leq \|C_\times\|_\infty + \mu_M (\|I\|_\infty \| [m']_{F_M} \|_\infty + \|I'\|_\infty \| [m]_{F_M} \|_\infty) \\ &\quad + b \mu_M^2 \|I\|_\infty \|I'\|_\infty. \end{aligned}$$

Moreover,

$$\begin{aligned} \|e_\times\|_\infty &\leq \mu_M (\| [m]_{F_M} \|_\infty + b \mu_M \|I\|_\infty) \|e'\|_\infty \\ &\quad + \mu_M (\| [m']_{F_M} \|_\infty + b \mu_M \|I'\|_\infty) \|e\|_\infty + \frac{\mu_M}{\Delta} \|e\|_\infty \|e'\|_\infty. \end{aligned}$$

For $M = b^n$ with $F_M(X) = X^n - X + b$, we have $\kappa = 0$ and may take $\mu_M = (b+1)n$. The special form of F_M also gives the sharper bounds $\|ta\|_\infty \leq 2b\|a\|_\infty$, $\| [m]_{F_M} \|_\infty \leq b-1$, $\|C_+\|_\infty \leq 1$, and $\|C_\times\|_\infty \leq n(b^2-1)$.

Since $\lfloor N/d \rfloor$ messages are packed into one CKKS ciphertext, each homomorphic arithmetic operation is applied to all packed messages simultaneously. The computation remains leveled until the approximation error or the overflow polynomial becomes too large, at which point we apply bootstrapping.

Comparison with the triangle representation For modulus 2^n , Gao and Zheng [20] represent a message m as $f = t^{-1}[m]_t + I \in \mathcal{P}_{2^n, \mathbb{R}}$. Multiplying by t gives the corresponding arithmetic polynomial $A = tf = [m]_t + tI$.

For two triangle representations f and f' , multiplication requires computing $tf f'$, which consumes two CKKS levels, whereas multiplication in our representation consumes only one. Both representations yield the same overflow polynomial $I_\times = C_\times + I[m']_t + I'[m]_t + tII'$ and therefore have the same overflow growth. Our representation additionally avoids the multiplication by t after the ciphertext multiplication, resulting in a smaller approximation error growth.

The numerical behavior of both the CKKS packing and homomorphic arithmetic depends on the defining polynomial F_M . We next describe how we choose F_M for a given modulus M .

3.4 Choosing the Defining Polynomial

The condition $F_M(b) = M$ is sufficient for the algebraic representation, while the choice of F_M determines the numerical behavior of its CKKS instantiation.

We require F_M to have d distinct nonreal roots and seek defining polynomials for which the transformations between coefficients and root evaluations are well conditioned and arithmetic modulo F_M has small coefficient growth. Although an algebraic defining polynomial exists for every modulus M , an arbitrary choice need not satisfy these numerical and arithmetic requirements. We therefore search for suitable defining polynomials using a lattice formulation and a closest-vector problem.

Lattice of defining polynomials. We choose an even degree d and a monic reference polynomial $G(X)$ with well-separated roots on the unit circle, such as $G_0(X) = X^d + 1$ or $G_1(X) = 1 + X + \dots + X^d$. Choose $E_0(X) \in \mathbb{Z}[X]$ with $\deg E_0 < d$ and $E_0(b) = M - G(b)$. Then every monic degree- d polynomial satisfying $F_M(b) = M$ can be written uniquely as

$$F_M(X) = G(X) + E_0(X) + (X - b)H(X),$$

where $H(X) = \sum_{j=0}^{d-2} h_j X^j$ with $h_j \in \mathbb{Z}$. Indeed, $F_M - (G + E_0)$ has degree at most $d - 1$ and vanishes at $X = b$, and is therefore divisible by $X - b$ over $\mathbb{Z}[X]$. Hence, for fixed (M, b, d) , all defining polynomials form an affine integer lattice generated by $X - b, X(X - b), \dots, X^{d-2}(X - b)$. The choice of G affects the search metric but not this set of defining polynomials.

Closest-vector search. Let $\omega_0, \dots, \omega_{d-1}$ be the roots of the reference polynomial G . Writing $H(X) = \sum_{j=0}^{d-2} h_j X^j$, we have $F_M(\omega_k) = E_0(\omega_k) + (\omega_k - b) \sum_{j=0}^{d-2} h_j \omega_k^j$. Hence, the evaluations of F_M at the roots of G depend linearly on the integer coefficient vector $\mathbf{h}$ of H .

Define $A \in \mathbb{C}^{d \times (d-1)}$ and $\mathbf{y} \in \mathbb{C}^d$ by $A_{k,j} = (\omega_k - b) \omega_k^j$ and $y_k = -E_0(\omega_k)$, and let $\tilde{A} \in \mathbb{R}^{2d \times (d-1)}$ and $\tilde{\mathbf{y}} \in \mathbb{R}^{2d}$ denote their real embeddings. The search problem is

$$\min_{\mathbf{h} \in \mathbb{Z}^{d-1}} \left\| \tilde{A} \mathbf{h} - \tilde{\mathbf{y}} \right\|_2,$$

which is a closest-vector problem in the lattice $\tilde{A} \mathbb{Z}^{d-1}$ with target $\tilde{\mathbf{y}}$. Minimizing this distance minimizes the values of F_M at the roots of G and favors defining polynomials that remain close to the reference polynomial at these points.

Lattice enumeration. We enumerate lattice vectors near the target under the CVP metric above. For each resulting polynomial, we verify that its roots are distinct and nonreal and evaluate the conditioning of the coefficient-to-root transformations and the coefficient growth under arithmetic modulo F_M . We retain candidates satisfying the numerical and arithmetic bounds required by the target homomorphic parameter set. The detailed enumeration procedure, candidate tests, certified bounds, and stopping criterion are given in Appendix B.1.

We next give the defining polynomials used in our concrete instantiations.

Moduli of the form b^n . For $M = b^n$ with even n , we take $d = n$ and $G(X) = X^n + 1$. This gives $F_{b^n}(X) = X^n - X + b$, which has n distinct nonreal roots and satisfies the arithmetic-growth bounds given above. Here $\kappa = 0$, so for $m = \sum_{i=0}^{n-1} m_i b^i$ with $m_i \in \{0, \dots, b-1\}$, we have $[m]_{F_{b^n}}(X) = \sum_{i=0}^{n-1} m_i X^i$. We write this representative simply as $[m]_t$ in the remainder of the paper.

We instantiate the construction for the field moduli of NIST P-384 [28] and Curve25519 [6].

P-384. For $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$, we consider $b = 2, 4, 8$, and 16 , corresponding to 1-, 2-, 3-, and 4-bit radix- b digits, respectively. The corresponding degrees are 384, 192, 128, and 96. Using $G(X) = X^d + 1$, the search gives the defining polynomials $X^{384} - X^{128} - X^{96} + X^{32} - X^2 + 3$, $X^{192} - X^{64} - X^{48} + X^{16} - X + 3$, $X^{128} - 4X^{42} - X^{32} + 4X^{10} - X + 7$, and $X^{96} - X^{32} - X^{24} + X^8 - X + 15$, respectively. Each polynomial evaluates to p_{384} at the corresponding value of b .

For all four parameter choices, $p_{384} < b^d$, so $\kappa = 0$ and the coefficients of $[m]_{F_M}$ are the radix- b digits of the message.

Curve25519. For $p_{25519} = 2^{255} - 19$, we consider $b = 2, 4, 8$, and 16 , with degrees 254, 128, 86, and 64, respectively. For $b = 2$, we use $F_{25519}(X) = \sum_{i=0}^{254} X^i - X^4 - X^2 + 2$, which satisfies $F_{25519}(2) = p_{25519}$. Since $2^{254} < p_{25519} < 2^{255}$, we have $\kappa = \lfloor m/2^{254} \rfloor \in \{0, 1\}$ and use $[m]_{F_{25519}}(X) = \sum_{i=0}^{253} m_i X^i - \kappa(F_{25519}(X) - X^{254})$.

For $b = 4, 8$, and 16 , using $G(X) = X^d + 1$, the search gives $X^{128} - 3X^{127} + 5X^{126} - 5X^{125} + 4X^{124} - X^2 - 2X + 5$, $X^{86} - 11X^{85} + 32X^{84} - 3X + 5$, and $X^{64} - 10X^{63} + 32X^{62} - 2X + 13$, respectively. For these three choices, $p_{25519} < b^d$, so $\kappa = 0$ and the coefficients of $[m]_{F_M}$ are the radix- b digits of the message.

The root properties and arithmetic-growth bounds of these defining polynomials are given in Appendix B.2.

4 Bootstrapping and Arbitrary Modular Arithmetic

Homomorphic arithmetic increases both the overflow polynomial and the CKKS approximation error. We use `OverflowRefresh` to replace the overflow polynomial with a bounded one and `ErrorRefresh` to reduce the approximation error without changing the arithmetic polynomial. The two procedures can be applied independently or combined into a complete arithmetic bootstrapping procedure.

As described in Section 3.2, one arithmetic ciphertext packs $\lfloor N/d \rfloor$ messages, and the same transformations are applied to all of them simultaneously. We therefore describe the procedures for one packed message.

4.1 Overflow Refresh

The overflow refresh exploits the fact that the overflow polynomial I has integer coefficients. For $A = [m]_{F_M} + tI$, multiplication by t^{-1} gives $t^{-1}A = t^{-1}[m]_{F_M} + I$, so the overflow polynomial becomes an additive integer polynomial that can be

removed modulo a small modulus. Since $F_M(b) = M$, the polynomial $F_M(X) - M$ is divisible by $X - b$. In $\mathcal{P}_{M,\mathbb{R}}$, we have $F_M(X) = 0$, and hence

$$t^{-1} = -\frac{1}{M} \frac{F_M(X) - M}{X - b}.$$

Under the root embedding, multiplication by t^{-1} corresponds to componentwise multiplication by $\rho(t^{-1}) = ((\xi_0 - b)^{-1}, \dots, (\xi_{d/2-1} - b)^{-1})^\top$. We absorb this multiplication into $\text{EvalToCoeff}_{\mathcal{P}_M}$ by replacing $\mathbf{V}_{\rho,h}$ with $\mathbf{V}_{\rho,h} \text{diag}(\rho(t^{-1}))$, and denote the resulting transformation by $\text{EvalToCoeff}_{\mathcal{P}_M}^{(t^{-1})}$. Its two output ciphertexts contain the two halves of the coefficient vector of $t^{-1}A$.

We then apply $\text{StC}_{\mathcal{R}}$ to place these coefficient values in an RLWE plaintext polynomial representing $\Delta(t^{-1}[m]_{F_M} + I)$ up to approximation error. The operation $\text{Truncate}_{\Delta \rightarrow q_0}$ removes the integer polynomial I modulo q_0 and may introduce a small integer polynomial I' . After $\text{ModRaise}_{q_0 \rightarrow Q}$, which introduces another bounded integer polynomial I'' , the plaintext represents $t^{-1}[m]_{F_M} + I' + I''$ at scale q_0 up to approximation error.

Finally, we absorb the multiplication by t and the factor Δ/q_0 into $\text{CoeffToEval}_{\mathcal{P}_M}$ by replacing $\mathbf{U}_{\rho,h}$ with $(\Delta/q_0) \text{diag}(\rho(t)) \mathbf{U}_{\rho,h}$. We denote the resulting transformation by $\text{CoeffToEval}_{\mathcal{P}_M}^{(t)}$, whose output represents $[m]_{F_M} + tI_{\text{out}}$ at scale Δ . These modifications only rescale the transformation matrices and do not change their nonzero diagonals, so the same rotations and Galois keys can be used.

We denote this procedure by OverflowRefresh , with flow $\text{EvalToCoeff}_{\mathcal{P}_M}^{(t^{-1})} \rightarrow \text{StC}_{\mathcal{R}} \rightarrow \text{Truncate}_{\Delta \rightarrow q_0} \rightarrow \text{ModRaise}_{q_0 \rightarrow Q} \rightarrow \text{CtS}_{\mathcal{R}} \rightarrow \text{CoeffToEval}_{\mathcal{P}_M}^{(t)}$. Its output overflow bound is given in Theorem 4. The proofs of the three correctness theorems in this section are given in Appendix A.2.

Theorem 4 (Correctness of OverflowRefresh). *For each packed message m , OverflowRefresh preserves m . Let e_c denote the coefficient error immediately before $\text{Truncate}_{\Delta \rightarrow q_0}$. Then, except with negligible probability, the output overflow polynomial satisfies $\|I_{\text{out}}\|_{\infty} \leq 15 + \lfloor \|t^{-1}[m]_{F_M}\|_{\infty} + \|e_c\|_{\infty} / \Delta + 1/2 \rfloor$. In particular, this bound is independent of the input overflow polynomial.*

For $M = b^n$, we take $F_M(X) = X^n - X + b$ and $t = X - b$. Writing $m = \sum_{i=0}^{n-1} m_i b^i$, the coefficients of $t^{-1}[m]_{F_M}$ are $u_0 = m/b^n - m_0/b$ and $u_j = -\sum_{i=0}^j m_i / b^{j-i+1}$ for $1 \leq j < n$. Since $0 \leq m_i < b$, we have $|u_j| < 1$ for every j , and therefore $\|t^{-1}[m]_{F_M}\|_{\infty} < 1$. If $\|e_c\|_{\infty} / \Delta < 1/2$, it follows that $\|I'\|_{\infty} \leq 1$ and hence $\|I_{\text{out}}\|_{\infty} \leq 16$.

4.2 Error Refresh

The error refresh reduces the approximation error without changing the arithmetic polynomial, following Meta-BTS [3]. For one packed message, let the input decrypt to $\Delta A + e$, where $A = [m]_{F_M} + tI$ has integer coefficients.

We apply $\text{EvalToCoeff}_{\mathcal{P}_M}$ and $\text{StC}_{\mathcal{R}}$ to place the coefficients of A in the RLWE plaintext polynomial. Then $\text{Truncate}_{\Delta \rightarrow q_0}$ removes the integer polynomial A ,

$\text{ModRaise}_{q_0 \rightarrow Q}$ raises the ciphertext modulus, and $\text{CtS}_{\mathcal{R}}$ followed by EvalMod_{q_0} recovers the approximation error. We map the recovered error back to the arithmetic slot representation using $\text{CoeffToEval}_{\mathcal{P}_M}^{(\Delta/q_0)}$, which includes the factor Δ/q_0 , and subtract it from the input ciphertext. The arithmetic polynomial A is therefore preserved, while its approximation error is replaced by the smaller error introduced by the refresh procedure. We denote this procedure by ErrorRefresh . Its error bound is given in Theorem 5.

Theorem 5 (Correctness of ErrorRefresh). *Let ε_c bound the additional error before $\text{Truncate}_{\Delta \rightarrow q_0}$ and ε_m the additional error before EvalMod_{q_0} . Let B_{mod} be the maximum input error supported by EvalMod_{q_0} . If $\|e\|_{\infty} + \varepsilon_c < \Delta/2$ and $(q_0/\Delta)(\|e\|_{\infty} + \varepsilon_c) + \varepsilon_m < B_{\text{mod}}$, then ErrorRefresh preserves A and returns a ciphertext decrypting to $\Delta A + e_{\text{out}}$, where*

$$\|e_{\text{out}}\|_{\infty} \leq \|e_A\|_{\infty} + \frac{2\pi^2}{3} \Delta \left(\frac{\|e\|_{\infty} + \varepsilon_c}{\Delta} + \frac{\varepsilon_m}{q_0} \right)^3.$$

Here, e_A bounds the remaining approximation and homomorphic evaluation errors.

4.3 Complete Arithmetic Bootstrapping

A complete arithmetic bootstrapping can be obtained by the direct composition $\text{ErrorRefresh} \circ \text{OverflowRefresh}$. The two procedures can also be fused to avoid repeating their intermediate transformations.

After $\text{ModRaise}_{q_0 \rightarrow Q}$ and $\text{CtS}_{\mathcal{R}}$ in OverflowRefresh , the CKKS slots contain the coefficients of $q_0(t^{-1}[m]_{F_M} + I_{\text{out}})$ up to approximation error. Instead of returning to the arithmetic representation, we directly multiply the corresponding coefficient polynomial by t modulo F_M . The slots then contain the coefficients of $q_0([m]_{F_M} + tI_{\text{out}})$ up to approximation error.

Since $[m]_{F_M} + tI_{\text{out}}$ has integer coefficients, we apply EvalRound_{q_0} [25] to remove the approximation error while preserving the integer coefficients. Finally, $\text{CoeffToEval}_{\mathcal{P}_M}^{(\Delta/q_0)}$ returns to the arithmetic slot representation and restores the scale to Δ .

We denote the fused procedure by ArithBootstrap , with flow $\text{EvalToCoeff}_{\mathcal{P}_M}^{(t^{-1})} \rightarrow \text{StC}_{\mathcal{R}} \rightarrow \text{Truncate}_{\Delta \rightarrow q_0} \rightarrow \text{ModRaise}_{q_0 \rightarrow Q} \rightarrow \text{CtS}_{\mathcal{R}} \xrightarrow{\times t} \text{EvalRound}_{q_0} \rightarrow \text{CoeffToEval}_{\mathcal{P}_M}^{(\Delta/q_0)}$. Compared with the direct composition, the fused procedure removes the intermediate $\text{CoeffToEval}_{\mathcal{P}_M}^{(t)}$ and $\text{EvalToCoeff}_{\mathcal{P}_M}$, as well as the second $\text{StC}_{\mathcal{R}}$, Truncate , ModRaise , and $\text{CtS}_{\mathcal{R}}$.

Theorem 6 (Correctness of ArithBootstrap). *Assume that the conditions of Theorem 4 hold. Let e_r be the approximation error entering EvalRound_{q_0} , and let B_{round} be the maximum error magnitude supported by EvalRound_{q_0} . If $\|e_r\|_{\infty} < B_{\text{round}}$, then ArithBootstrap preserves m and returns a ciphertext decrypting to $\Delta([m]_{F_M} + tI_{\text{out}}) + e_{\text{out}}$, where I_{out} satisfies the bound of Theorem 4.*

4.4 Arbitrary Modular Arithmetic

For a modulus M for which no suitable defining polynomial F_M is available, we perform Montgomery multiplication over $\mathcal{P}_{b^n}$ and $\mathcal{P}_{b^{2n}}$. We choose a base b with $\gcd(b, M) = 1$ and an even n satisfying $b^n > 34^2 M$. The two arithmetic spaces are defined by $X^n - X + b$ and $X^{2n} - X + b$, respectively. For a public integer $0 \leq a < b^n$ with base b expansion $a = \sum_i a_i b^i$, let $\text{Rad}_b(a) = \sum_i a_i X^i$. We use these polynomials for the public values M and $-M^{-1} \bmod b^n$ instead of multiplying by the corresponding large integers.

We represent $m \in \mathbb{Z}_M$ in $\mathcal{P}_{b^{2n}}$ by an arithmetic polynomial $[\tilde{m}]_{F_{b^{2n}}} + tI$, where $0 \leq \tilde{m} < 34M$, $\tilde{m} \equiv mb^n \pmod{M}$, $\|I\|_\infty \leq 16$, and $\deg I \leq n - 2$. Since $34M < b^n$, the canonical polynomial $[\tilde{m}]_{F_{b^{2n}}}$ is its base b polynomial and has degree less than n . The term tI also has degree less than n . For two input representations, let $P(X) = ([\tilde{m}_1]_{F_{b^{2n}}} + tI_1)([\tilde{m}_2]_{F_{b^{2n}}} + tI_2)$. Then $\deg P \leq 2n - 2$, so no reduction by $X^{2n} - X + b$ occurs and $P(b) = \tilde{m}_1 \tilde{m}_2$.

We first compute the Montgomery quotient modulo b^n . Let $q \in [0, b^n)$ be the unique integer satisfying $\tilde{m}_1 \tilde{m}_2 + qM \equiv 0 \pmod{b^n}$. We retain the lower n coefficients of $P(X)$. The omitted terms evaluate at $X = b$ to multiples of b^n , so the retained polynomial represents $\tilde{m}_1 \tilde{m}_2$ modulo b^n . We multiply it by $\text{Rad}_b(-M^{-1} \bmod b^n)$ and again retain only the lower n coefficients. Let $Q(X)$ denote the resulting polynomial of degree less than n . By construction, $Q(b) \equiv q \pmod{b^n}$.

We regard $Q(X)$ as an arithmetic polynomial in $\mathcal{P}_{b^n}$ and apply `ArithBootstrap`. The refreshed polynomial represents the same residue q and has the form $[q]_{F_{b^n}} + tJ$, where $\|J\|_\infty \leq 16$. Let $h = J_{n-1}$, so $|h| \leq 16$. When this element is written as its polynomial representative of degree less than n , the degree n term hX^n of tJ is removed using $X^n - X + b$. Since $(X^n - X + b)(b) = b^n$, the resulting polynomial evaluates at $X = b$ to $q - hb^n$.

We place this polynomial in $\mathcal{P}_{b^{2n}}$ and multiply it by $\text{Rad}_b(M)$. Both polynomials have degree less than n , so their product has degree at most $2n - 2$ and no reduction by $X^{2n} - X + b$ occurs. We then add the original product $P(X)$ and the public polynomial $16X^n \text{Rad}_b(M)$. At $X = b$, the resulting polynomial evaluates to

$$\tilde{m}_1 \tilde{m}_2 + qM + (16 - h)Mb^n.$$

The first two terms are divisible by b^n , and hence the whole value is divisible by b^n . Dividing by b^n gives

$$\tilde{m}' = \frac{\tilde{m}_1 \tilde{m}_2 + qM}{b^n} + (16 - h)M.$$

The same redundant range is preserved. Since $\tilde{m}_i < 34M$ and $b^n > 34^2 M$, we have $\tilde{m}_1 \tilde{m}_2 / b^n < M$. Moreover, $q < b^n$ gives $qM / b^n < M$, while $|h| \leq 16$ gives $0 \leq 16 - h \leq 32$. Therefore, $0 \leq \tilde{m}' < 34M$. The congruences $\tilde{m}_i \equiv m_i b^n \pmod{M}$ also give $\tilde{m}' \equiv m_1 m_2 b^n \pmod{M}$. Thus, the output has the same Montgomery representation as the inputs.

Finally, $\tilde{m}' < 34M < b^n$ implies that the value before division is $b^n \tilde{m}' < b^{2n}$. Its canonical polynomial in $\mathcal{P}_{b^{2n}}$ is therefore $X^n [\tilde{m}']_{F_{b^{2n}}}$. Hence, division

by b^n amounts to mapping the upper n canonical coefficients to the lower n positions. We incorporate this transformation into the final stage of a second **ArithBootstrap** in $\mathcal{P}_{b^{2n}}$. This bootstrap also refreshes the approximation error and polynomial overflow and returns a valid arithmetic representation $[\tilde{m}']_{F_{b^{2n}}} + tI'$. The parameters are chosen so that both bootstrap calls satisfy the correctness conditions of Section 4.

Algorithm 1 Montgomery multiplication over an arbitrary modulus M

Require: ct_1, ct_2 representing valid Montgomery arithmetic polynomials in $\mathcal{P}_{b^{2n}}$

Ensure: ct_{out} representing their product modulo M

- 1: $\text{ct}_{\text{prod}} \leftarrow \text{CMult}(\text{ct}_1, \text{ct}_2)$
 - 2: Retain the lower n coefficients of ct_{prod}
 - 3: Multiply by $\text{Rad}_b(-M^{-1} \bmod b^n)$ and retain the lower n coefficients again to obtain $Q(X)$
 - 4: Apply **ArithBootstrap** to $Q(X)$ in $\mathcal{P}_{b^n}$
 - 5: Write the refreshed quotient as a polynomial of degree less than n and place it in $\mathcal{P}_{b^{2n}}$
 - 6: Multiply it by $\text{Rad}_b(M)$ and add $\text{ct}_{\text{prod}} + 16X^n \text{Rad}_b(M)$
 - 7: Apply **ArithBootstrap** with the final transformation corresponding to division by b^n
 - 8: **return** ct_{out}
-

5 Arithmetic–Digit Conversion

In this section, we give conversions between the arithmetic representation and radix digit representations. We first define the digit representation and its linear conversion to the arithmetic representation. We then give arithmetic-to-digit conversion over an arbitrary integer modulus using exactly two functional bootstrap calls, followed by a batched variant. Finally, we specialize the conversion to binary output over $\mathbb{Z}_{2^n}$ and obtain a SIMD arithmetic logic unit (ALU).

5.1 Digit Representation and Digit-to-Arithmetic Conversion

Let $M \geq 2$, let $b \geq 2$, and let $\mathcal{P}_M = \mathbb{Z}[X]/\langle F_M(X) \rangle$, where $F_M(b) = M$. For a radix $B = b^s$, write the canonical radix representation of m as $m = \sum_j d_j B^j$, where $0 \leq d_j < B$. The digit representation stores these radix digits directly in CKKS slots.

The digit-to-arithmetic conversion requires only a linear transformation. The radix digits define the polynomial $\sum_j d_j X^{sj} \bmod F_M(X)$, whose value at $X = b$ is congruent to $\sum_j d_j b^{sj} = m$ modulo M . Since each polynomial $X^{sj} \bmod F_M(X)$ is public, the radix digits can be mapped linearly to the root evaluations of this polynomial and hence to the arithmetic representation.

5.2 Arithmetic-to-Digit Conversion

Recall that the representative in Equation 1 writes $m = m_0 + \kappa b^d$, where $0 \leq m_0 < b^d$ and $\kappa = \lfloor m/b^d \rfloor$. Let $B = b^s$ with $s \geq 1$, let $\ell = \lceil d/s \rceil$, and let $h = d - s(\ell - 1)$. Writing $m_0 = \sum_{i=0}^{d-1} m_i b^i$, we group its radix- b digits into radix- B digits $d_0, \dots, d_{\ell-1}$.

From Coefficients to Radix Values. As in OverflowRefresh, we first replace I by a bounded integer polynomial I' and obtain the coefficients of $t^{-1}[m]_{F_M} + I'$ in CKKS slots. We then apply a linear transformation that multiplies by $t = X - b$ modulo F_M and combines every s consecutive coefficients with weights $1, b, \dots, b^{s-1}$. Let $A = [m]_{F_M} + tI' = \sum_{i=0}^{d-1} A_i X^i$ and write $F_M(X) = X^d + \sum_{i=0}^{d-1} f_i X^i$. Since $[m]_{F_M}$ contributes $m_i - \kappa f_i$ and the reduction of the degree- d term $I'_{d-1} X^d$ contributes $-I'_{d-1} f_i$, we have $A_i = m_i + I'_{i-1} - bI'_i - \lambda f_i$, where $\lambda = \kappa + I'_{d-1}$ and $I'_{-1} = 0$.

For $0 \leq j < \ell - 1$, define $x_j = \sum_{r=0}^{s-1} b^r A_{sj+r}$ and $\alpha_j = \sum_{r=0}^{s-1} b^r f_{sj+r}$, and define $x_{\ell-1}$ and $\alpha_{\ell-1}$ analogously using the remaining h coefficients. Since $d_j = \sum_{r=0}^{s-1} b^r m_{sj+r}$ and the weighted overflow terms cancel except at the two boundaries, we obtain

$$x_j = d_j + I'_{sj-1} - bI'_{s(j+1)-1} - \lambda \alpha_j, \quad x_{\ell-1} = d_{\ell-1} + I'_{s(\ell-1)-1} - b^h I'_{d-1} - \lambda \alpha_{\ell-1}.$$

Thus, each radix value contains the corresponding digit, two boundary overflow terms, and the bounded term determined by λ .

Eliminating the Overflow Contribution. For $0 \leq j < \ell - 1$, choose a representative r_j of x_j modulo B and write $x_j = r_j + Bq_j$; for the last position, write $x_{\ell-1} = r_{\ell-1} + b^h q_{\ell-1}$. We combine the overflow refresh, radix formation, and modular reduction into one functional bootstrap $\text{FBS}_B^{(1)}$, which outputs the remainders r_j . The quotients are then obtained linearly as $q_j = (x_j - r_j)/B$ for $j < \ell - 1$ and $q_{\ell-1} = (x_{\ell-1} - r_{\ell-1})/b^h$.

For each $0 \leq j < \ell - 1$, there is a unique carry $c_{j+1} \in \mathcal{C}$ such that

$$r_j = d_j + I'_{sj-1} - \lambda \alpha_j - Bc_{j+1}.$$

For the last position, define c_ℓ analogously by $r_{\ell-1} = d_{\ell-1} + I'_{s(\ell-1)-1} - \lambda \alpha_{\ell-1} - b^h c_\ell$. We set $c_0 = 0$ and let $\mathcal{C}$ denote the public finite set containing all possible carry values. Comparing these relations with the expressions for x_j gives $q_j = c_{j+1} - I'_{s(j+1)-1}$ for $j < \ell - 1$ and $q_{\ell-1} = c_\ell - I'_{d-1}$.

We then compute $a_0 = r_0$ and $a_j = r_j + q_{j-1}$ for $1 \leq j < \ell$ by a linear transformation. Substituting the expressions for r_j and q_{j-1} cancels the boundary coefficient I'_{sj-1} and gives

$$a_j = d_j + c_j - Bc_{j+1} - \lambda \alpha_j, \quad 0 \leq j < \ell - 1. \quad (2)$$

For the last radix position, we similarly obtain $a_{\ell-1} = d_{\ell-1} + c_{\ell-1} - b^h c_\ell - \lambda \alpha_{\ell-1}$.

Carry Functions. Since κ and I'_{d-1} lie in public bounded ranges, $\lambda = \kappa + I'_{d-1}$ ranges over a finite set of candidates. Fix a candidate λ and a possible current carry $c \in \mathcal{C}$. For $j < \ell - 1$, Equation 2 becomes $a_j = d_j + c - Bc_{j+1} - \lambda\alpha_j$. Since $0 \leq d_j < B$, this uniquely determines the next carry as

$$f_{j,\lambda}(c) = \left\lceil \frac{c - a_j - \lambda\alpha_j}{B} \right\rceil. \quad (3)$$

The last radix position is handled in the same way with radix b^h . For each $c' \in \mathcal{C}$, the indicator $\mathbf{1}[f_{j,\lambda}(c) = c']$ specifies whether the next carry is c' when the current carry is c . For the actual value of λ and $c = c_j$, we have $f_{j,\lambda}(c_j) = c_{j+1}$.

The final quotient is used to check each candidate λ . If the final carry for a candidate is c , then $q_{\ell-1} = c - I'_{d-1}$ gives $I'_{d-1} = c - q_{\ell-1}$ and $\kappa = \lambda - c + q_{\ell-1}$. Therefore, the candidate can pass only if $c - q_{\ell-1}$ satisfies the bound on I'_{d-1} and $0 \leq \lambda - c + q_{\ell-1} \leq \lfloor (M-1)/b^d \rfloor$.

The carry indicators are functions of a_j , while the candidate check and the radix- b digits of κ are functions of $q_{\ell-1}$. We compute these functions together using a multi-value bootstrap, denoted by $\text{FBS}_B^{(2)}$. Its input places $a_0, \dots, a_{\ell-1}$ in slots $0, \dots, \ell-1$ and $q_{\ell-1}$ in slot ℓ . For each (λ, c, c') , $\text{FBS}_B^{(2)}$ outputs a ciphertext $\text{ct}_{\lambda,c,c'}^f$ whose slot j contains $\mathbf{1}[f_{j,\lambda}(c) = c']$ for $0 \leq j < \ell$. For each (λ, c) , it also outputs a ciphertext $\text{ct}_{\lambda,c}^{\text{chk}}$ whose slot ℓ contains the bit indicating whether $I'_{d-1} = c - q_{\ell-1}$ and $\kappa = \lambda - c + q_{\ell-1}$ satisfy their required bounds. When $M > b^d$, for each radix- b digit position r of κ , it additionally outputs a ciphertext $\text{ct}_{\lambda,c,r}^\kappa$ whose slot ℓ contains the corresponding digit of the candidate κ . When $M \leq b^d$, we have $\kappa = 0$, so these ciphertexts are unnecessary and the candidate check reduces to $\lambda = c - q_{\ell-1}$.

Logarithmic Carry Computation. Computing the carries one radix position at a time would require multiplicative depth linear in ℓ . Instead, we compose the carry functions over consecutive radix positions. For a fixed λ and $0 \leq i \leq j \leq \ell-1$, let $f_{i:j,\lambda} = f_{j,\lambda} \circ \dots \circ f_{i,\lambda}$. If $c_i = c$, then $f_{i:j,\lambda}(c) = c_{j+1}$.

To compose the carry functions, fix $i \leq m < j$. The positions $i, \dots, m$ map $c_i = c$ to the intermediate carry c_{m+1} , and the positions $m+1, \dots, j$ map this carry to c_{j+1} . Writing c' and c'' for possible values of c_{m+1} and c_{j+1} , respectively, we obtain

$$\mathbf{1}[f_{i:j,\lambda}(c) = c''] = \sum_{c' \in \mathcal{C}} \mathbf{1}[f_{i:m,\lambda}(c) = c'] \mathbf{1}[f_{m+1:j,\lambda}(c') = c'']. \quad (4)$$

Exactly one value of c' equals the carry c_{m+1} produced from $c_i = c$, so the sum selects the corresponding next carry.

We evaluate Equation 4 homomorphically on the ciphertexts $\text{ct}_{\lambda,c,c'}^f$ using rotations and ciphertext multiplications. Each composition doubles the number of radix positions included in the carry function, so after $\lceil \log_2 \ell \rceil$ compositions, slot j of $\text{ct}_{\lambda,0,c}^f$ contains $\mathbf{1}[f_{0:j,\lambda}(0) = c]$. For each candidate λ , we therefore obtain

the candidate carries

$$c_{0,\lambda} = 0, \quad c_{j,\lambda} = \sum_{c \in \mathcal{C}} c \mathbf{1}[f_{0:j-1,\lambda}(0) = c], \quad 1 \leq j \leq \ell.$$

For the actual value λ^* , these are exactly the carries $c_0, \dots, c_\ell$.

Selecting the Correct Value of λ . For each candidate λ , we combine the final-carry indicators with the check bits produced by $\text{FBS}_B^{(2)}$ to obtain an encrypted bit

$$b_\lambda = \sum_{c \in \mathcal{C}} \mathbf{1}[c_{\ell,\lambda} = c] \mathbf{1} \left[\begin{array}{l} c - q_{\ell-1} \text{ satisfies the bound on } I'_{d-1}, \\ 0 \leq \lambda - c + q_{\ell-1} \leq \lfloor (M-1)/b^d \rfloor \end{array} \right].$$

Thus, $b_\lambda = 1$ exactly when $I'_{d-1} = c_{\ell,\lambda} - q_{\ell-1}$ and $\kappa_\lambda = \lambda - c_{\ell,\lambda} + q_{\ell-1}$ satisfy their required bounds.

More than one candidate may satisfy $b_\lambda = 1$. Let λ^* be the actual value of λ . For the proof, define $d_{j,\lambda} = a_j - c_{j,\lambda} + Bc_{j+1,\lambda} + \lambda\alpha_j$ for $j < \ell - 1$ and define the last digit analogously with radix b^h . Let $m_{0,\lambda}$ be the integer represented by these digits. Summing Equation 2 with weights b^{sj} gives

$$m_{0,\lambda} + \kappa_\lambda b^d = m + (\lambda - \lambda^*)M. \quad (5)$$

If $\lambda < \lambda^*$, the right-hand side is at most $m - M < 0$, whereas $b_\lambda = 1$ implies $m_{0,\lambda} \geq 0$ and $\kappa_\lambda \geq 0$, a contradiction. Since λ^* itself satisfies the check, it is exactly the smallest candidate with $b_\lambda = 1$.

Let the candidates be ordered as $\lambda_1 < \dots < \lambda_t$, and let $b_i = b_{\lambda_i}$. The encrypted bit

$$b_i \prod_{r < i} (1 - b_r) \quad (6)$$

is 1 exactly for $\lambda_i = \lambda^*$. These bits select λ^* and the corresponding carries c_j in $O(\log t)$ multiplicative depth. The radix digits are then recovered linearly as

$$d_j = a_j - c_j + Bc_{j+1} + \lambda^* \alpha_j, \quad d_{\ell-1} = a_{\ell-1} - c_{\ell-1} + b^h c_\ell + \lambda^* \alpha_{\ell-1}.$$

When $M > b^d$, the same selection also gives the radix- b digits of κ produced by $\text{FBS}_B^{(2)}$; when $M \leq b^d$, we have $\kappa = 0$.

Thus, arithmetic-to-digit conversion uses exactly two functional bootstrap calls followed by $O(\log \ell + \log t)$ multiplicative depth. The detailed algorithm is given in Appendix C.1.

5.3 Batched Conversion

The conversion above already processes all messages packed in one arithmetic ciphertext. To process several arithmetic ciphertexts together, we place the radix values of each input in a disjoint slot interval. For the general conversion, the interval must contain the ℓ radix positions, one additional position for $q_{\ell-1}$, and

the $\lceil \log_B M \rceil$ output positions. The position for $q_{\ell-1}$ and the output positions are used at different stages and may overlap. We therefore use intervals of length $2^{\lceil \log_2 \max\{\ell+1, \lceil \log_B M \rceil\} \rceil}$. Since one packed message occupies $d/2$ CKKS slots, at most $\lfloor d/2^{1+\lceil \log_2 \max\{\ell+1, \lceil \log_B M \rceil\} \rceil} \rfloor$ arithmetic ciphertexts can be processed together.

Before $\text{FBS}_B^{(1)}$, the coefficient data from all inputs are arranged in these disjoint intervals, so the first functional bootstrap is shared by all inputs. Its radix outputs and last quotients are kept in the same intervals before $\text{FBS}_B^{(2)}$, so the second functional bootstrap, the logarithmic carry computation, the selection of λ , and the final digit recovery are also shared. After each rotation, plaintext masks remove values that cross interval boundaries and keep the computations for different inputs independent. Thus, batching reduces the amortized conversion cost without increasing the number of functional bootstrap calls or the multiplicative depth.

5.4 SIMD ALU with $O(1)$ bootstrapping

For $M = 2^n$, we take $b = 2$, $d = n$, $F_M(X) = X^n - X + 2$, and $t = X - 2$. The same construction applies to arithmetic bases that are powers of 2, with the corresponding degree and defining polynomial adjusted accordingly. Then $\kappa = 0$. For $B = 2^s$ with $s \geq 2$, the two nonzero lower coefficients of F_M cancel under the weights 1 and 2, so all coefficient sums α_j vanish. Equation 2 therefore reduces to

$$a_j = d_j + c_j - Bc_{j+1},$$

with the corresponding radix 2^h at the last position.

The digit and carry functions no longer depend on λ , so we omit this index and write the carry function as $f_j(c)$. To obtain Boolean output, $\text{FBS}_B^{(2)}$ evaluates

$$b_{j,k}(c) = \text{bit}_k(a_j - c + Bf_j(c))$$

for $0 \leq j < \ell - 1$, with B replaced by 2^h at the last radix position. For the actual carry $c = c_j$, we have $f_j(c_j) = c_{j+1}$, and Equation 2 gives $a_j - c_j + Bf_j(c_j) = d_j$. Hence, $b_{j,k}(c_j) = m_{sj+k}$ whenever $sj + k < n$. The carry computation is unchanged, and the same logarithmic composition selects the bit functions corresponding to the actual carries. Thus, arithmetic-to-Boolean conversion also uses exactly two functional bootstrap calls.

The Boolean representation uses two ciphertexts: the first contains the lower $n/2$ bits and the second contains the upper $n/2$ bits, with the same packing for every message. Boolean operations are evaluated directly on these bits, and the two ciphertexts can be refreshed independently using discrete CKKS bootstrapping [4]. Conversely, the Boolean-to-arithmetic conversion is the linear digit-to-arithmetic conversion above with radix 2. Together with leveled arithmetic and arithmetic bootstrapping in the arithmetic representation, these conversions give a SIMD ALU.

Algorithm 2 Arithmetic-to-Boolean conversion A2B

Require: ct_A encrypting $\rho([m]_t + tI)$

Ensure: $(\text{ct}_{B,0}, \text{ct}_{B,1})$ encrypting the lower and upper $n/2$ bits of m

- 1: $(\text{ct}_r, \text{ct}_q) \leftarrow \text{FBS}_B^{(1)}(\text{ct}_A)$ $\triangleright$ radix reduction
 - 2: $(\{\text{ct}_{c,c'}^f\}, \{\text{ct}_{c,k}^b\}) \leftarrow \text{FBS}_B^{(2)}(\text{LinTrans}_a(\text{ct}_r, \text{ct}_q))$ $\triangleright$ local functions
 - 3: **for** $\delta = 1, 2, 4, \dots < \ell$ **do**
 - 4: $\mu_\delta^L \leftarrow (\mathbf{1}[0 \leq j < \delta])_j, \quad \mu_\delta^R \leftarrow (\mathbf{1}[\delta \leq j < \ell])_j$
 - 5: $\hat{\text{ct}}_{c,u}^f \leftarrow \text{PMult}(\text{Rot}_{-\delta}(\text{ct}_{c,u}^f), \mu_\delta^R)$
 - 6: **simultaneously for all** $c, c' \in \mathcal{C}$ **and** $0 \leq k < s$:
 $\text{ct}_{c,k}^b \leftarrow \text{PMult}(\text{ct}_{c,k}^b, \mu_\delta^L) + \sum_{u \in \mathcal{C}} \text{CMult}(\hat{\text{ct}}_{c,u}^f, \text{ct}_{u,k}^b)$
 $\text{ct}_{c,c'}^f \leftarrow \text{PMult}(\text{ct}_{c,c'}^f, \mu_\delta^L) + \sum_{u \in \mathcal{C}} \text{CMult}(\hat{\text{ct}}_{c,u}^f, \text{ct}_{u,c'}^f)$
 - 7: **end for**
 - 8: **return** $\text{LinTrans}_{\text{out}}(\{\text{ct}_{0,k}^b\}_{0 \leq k < s})$ $\triangleright c_0 = 0$
-

5.5 Cost Analysis

We analyze the arithmetic-to-Boolean conversion over $\mathbb{Z}_{2^n}$ used in our SIMD ALU. The corresponding analysis for arithmetic-to-digit conversion over general moduli is given in Appendix C.2.

For the Boolean-output specialization, the conversion uses exactly two functional bootstrap calls, independent of the bit width n , and the carry computation requires $\lceil \log_2 \ell \rceil$ multiplicative levels, where $\ell = \lceil n/s \rceil$. For a carry set $\mathcal{C}$, one outgoing-carry indicator for each possible current carry can be omitted because the outgoing indicators sum to one. Thus, $\text{FBS}_B^{(2)}$ evaluates $|\mathcal{C}| - 1$ carry indicators and s bit functions for each value in $\mathcal{C}$, for a total of $|\mathcal{C}|(|\mathcal{C}| + s - 1)$ output functions. The complete carry computation uses

$$|\mathcal{C}|(|\mathcal{C}| - 1)(|\mathcal{C}| + s - 1)(\lceil \log_2 \ell \rceil - 1) + s(|\mathcal{C}| - 1)$$

ciphertext multiplications.

To instantiate these costs, we determine the size of $\mathcal{C}$ as a function of the radix B . By Theorem 4, $|I'_i| \leq 16$, so $d_j + I'_{s_j-1}$ ranges over $B + 32$ consecutive integers. Since one carry value can cover at most B of them, $|\mathcal{C}| \geq \lceil (B + 32)/B \rceil$, and this bound is achievable by an appropriate choice of remainder representatives. Hence, $|\mathcal{C}| = 9, 5, 3$ for $B = 4, 8, 16$, respectively, and $|\mathcal{C}| = 2$ for $B \geq 32$.

These bounds motivate the choice $B = 32$. For $B < 32$, the larger carry set increases both the number of bootstrap outputs and the carry-composition cost, whereas for $B > 32$ the carry set remains binary while the lookup cost increases with s . Thus, $B = 32$ is the smallest radix with binary carries and gives the best tradeoff among the considered choices. A detailed comparison is given in Appendix C.3.

For $B = 32$, we have $|\mathcal{C}| = 2$ and $s = 5$, so $\text{FBS}_B^{(2)}$ evaluates 12 output functions. Each nonfinal composition level requires 12 ciphertext-ciphertext

multiplications, while the final level requires only 5 because $c_0 = 0$ is known and only the output bits are needed.

We now consider the multiplicative depth of the $B = 32$ implementation. $\text{FBS}_B^{(2)}$ uses 2 levels for $\text{CtS}_{\mathcal{R}}$, 6 levels for the exponential approximation, 2 double-angle levels, 7 levels for the lookup, and 2 levels for $\text{StC}_{\mathcal{R}}$. These steps require 19 levels in total, and the input preparation does not add another level. Since $\ell = \lceil n/5 \rceil$, the total multiplicative depth is $19 + \lceil \log_2 \ell \rceil$.

6 Implementation and Performance

6.1 Experimental Setup

We implemented our construction in OpenFHE [2]. All experiments were run on one logical core of an Intel Xeon Gold 6248R processor at 3.00 GHz under Fedora 38. The code was compiled with GCC 13.2.1 using `-O3 -DNDEBUG`.

For an arithmetic ciphertext decrypting to $\Delta([m]_{F_M} + tI) + e$, we define the precision as $\log_2(\Delta/\|e\|_\infty)$, where e is the CKKS approximation error. The RLWE error is sampled from a Gaussian distribution with standard deviation $\sigma = 3.2$. We use the `HEStd_128_classic` security setting. Table 2 summarizes the parameters used in our arithmetic and representation-conversion experiments.

Table 2. Parameters used in the arithmetic and conversion experiments.

Task	N	$\log_2 \Delta$	σ	wt(sk)	wt(sk _{enc})	$\log_2 Q$	$\log_2 P$
Arithmetic	2^{16}	43	3.2	192	32	904	400
Conversion	2^{16}	43	3.2	192	32	$(44 + 43(19 + \lceil \log_2 \lceil n/5 \rceil \rceil))$ bits	400

For comparisons with CPL [13] and GZ [20], we use the parameter sets provided by their public implementations. For TFHE-rs [31], we use its default parameter set targeting 128-bit security. Unless stated otherwise, each reported latency is the arithmetic mean of three measured runs after one warmup. An em dash denotes an unavailable measurement.

6.2 Arithmetic Computation

We evaluate homomorphic integer multiplication using the direct arithmetic representation with arithmetic base $b = 2$. Each multiplication consists of one ciphertext multiplication followed by `ArithBootstrap`, so the output is again a refreshed arithmetic representation and can be used directly in subsequent multiplications. The arithmetic base can also be chosen differently: increasing b reduces the degree of the defining polynomial and hence increases the packing capacity. We use $b = 2$ throughout the direct arithmetic experiments.

For the RSA-1024 and RSA-2048 moduli, we use the Montgomery construction of Section 4.4 with $b = 16$, $n = 260$ and 516 , respectively, and a 52-bit scaling factor. The corresponding arithmetic spaces $\mathcal{P}_{b^{2n}}$ have degrees 520 and 1032 and pack 126 and 63 values, respectively, for $N = 2^{16}$. Each modular multiplication consists of one ciphertext multiplication and two arithmetic refreshes and returns to the same Montgomery representation.

The amortized time is the total online latency divided by the number of packed integers. CPL LazyMult performs multiplication followed by partial carry reduction, while CPL ExactMult additionally restores the canonical radix representation. TFHE-rs evaluates one scalar integer multiplication.

Table 3 reports the online latency, amortized time per packed integer, and output precision. Each Lat./Am. entry gives the online latency followed by the amortized time per packed integer, and Prec. denotes the precision of the refreshed output.

Table 3. Performance of homomorphic integer multiplication.

M	Ours			CPL			Speedup	
	Pack.	Lat./Am.	Prec.	Pack.	Lazy	Exact	Lazy	Exact
$\mathbb{Z}_{2^{32}}$	2048	29.11/0.01 s	19.11	2048	57.84/0.03 s	88.69/0.04 s	1.99×	3.05×
$\mathbb{Z}_{2^{64}}$	1024	27.37/0.03 s	20.06	1024	60.79/0.06 s	91.65/0.09 s	2.22×	3.35×
$\mathbb{Z}_{2^{128}}$	512	29.96/0.06 s	21.23	512	98.53/0.19 s	132.41/0.26 s	3.29×	4.42×
$\mathbb{Z}_{2^{256}}$	256	35.95/0.14 s	21.65	256	100.87/0.39 s	165.26/0.65 s	2.81×	4.60×
$\mathbb{Z}_{2^{512}}$	128	41.18/0.32 s	22.49	128	107.63/0.84 s	174.48/1.36 s	2.61×	4.24×
$\mathbb{Z}_{2^{1024}}$	64	55.95/0.87 s	23.24	64	117.40/1.83 s	186.11/2.91 s	2.10×	3.33×
$\mathbb{Z}_{p^{25519}}$	258	43.10/0.17 s	21.49	128	373.62/2.92 s	541.95/4.23 s	17.47×	25.35×
$\mathbb{Z}_{p^{384}}$	170	45.11/0.27 s	21.92	64	462.66/7.23 s	605.04/9.45 s	27.24×	35.63×
RSA-1024	126	202.94/1.61 s	18.53	32	—	862.93/26.97 s	—	16.74×
RSA-2048	63	213.54/3.39 s	18.61	16	—	1619.24/101.20 s	—	29.85×

Lat./Am. entries report online latency / amortized time. Prec. is measured in bits. Speedups are computed from the unrounded amortized times.

Arithmetic refresh. We evaluate OverflowRefresh and ErrorRefresh separately and compare their direct composition with the fused arithmetic bootstrapping procedure. Table 4 reports their running times for the same $b = 2$ arithmetic settings.

6.3 Representation Conversion

We evaluate the Boolean-output specialization of our arithmetic-to-digit conversion using arithmetic base $b = 2$ and radix $B = 32$. The conversion uses exactly two functional bootstrap calls followed by logarithmic-depth carry composition. For $n = 32, 64, 128$, and 256 , we use multiplicative depths 23, 24, 25, and 26, respectively. Table 5 reports the conversion time.

Table 4. Performance of arithmetic refreshing.

Procedure	$\mathbb{Z}_{2^{32}}$	$\mathbb{Z}_{2^{64}}$	$\mathbb{Z}_{2^{128}}$	$\mathbb{Z}_{2^{256}}$	$\mathbb{Z}_{2^{512}}$	$\mathbb{Z}_{2^{1024}}$	$\mathbb{Z}_{p_{25519}}$	$\mathbb{Z}_{p_{384}}$
OverflowRefresh	23.39 s	22.44 s	27.54 s	36.43 s	51.93 s	81.95 s	34.38 s	46.65 s
ErrorRefresh	28.01 s	26.74 s	29.66 s	36.99 s	42.43 s	59.87 s	36.68 s	44.73 s
Composed	52.14 s	49.31 s	57.85 s	73.92 s	95.06 s	141.65 s	72.52 s	86.45 s
Fused	28.76 s	27.02 s	29.61 s	35.60 s	40.83 s	55.60 s	42.75 s	44.76 s

Composed denotes $\text{ErrorRefresh} \circ \text{OverflowRefresh}$, while Fused denotes ArithBootstrap .

Table 5. Performance of representation conversion with Boolean output and $B = 32$. Bold values report speedups over GZ.

	$n = 32$	$n = 64$	$n = 128$	$n = 256$
GZ [20]	166.40 s	331.60 s	684.85 s	1718.04 s
Ours	82.04 s	88.48 s	107.09 s	121.28 s
	2.03×	3.75×	6.40×	14.17×

7 Conclusion

We presented a framework for efficient SIMD homomorphic integer computation over arbitrary plaintext moduli. Our arithmetic polynomial representation supports leveled addition and multiplication with one CKKS level per multiplication, together with arithmetic refresh for controlling approximation error and polynomial overflow. For moduli admitting suitable defining polynomials, we give a CVP-based method for finding such polynomials, while general moduli are handled through homomorphic Montgomery multiplication over fixed arithmetic spaces using only $O(1)$ bootstrap calls.

We further presented an arithmetic-to-digit conversion that supports different arithmetic bases and output radices. The conversion uses $O(1)$ functional bootstrap calls and logarithmic-depth carry computation, and directly yields arithmetic-to-Boolean conversion as a special case. Together, these techniques provide efficient arithmetic, representation conversion, and non-arithmetic computation within a unified SIMD framework.

References

1. Alexandru, A., Kim, A., Polyakov, Y.: General functional bootstrapping using CKKS. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part III. LNCS, vol. 16002, pp. 304–337. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01881-6_10
2. Badawi, A.A., Alexandru, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Pascoe, C., Polyakov, Y., Quah, I., R. V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: OpenFHE: Open-source fully

- homomorphic encryption library. Cryptology ePrint Archive, Report 2022/915 (2022), <https://eprint.iacr.org/2022/915>
3. Bae, Y., Cheon, J.H., Cho, W., Kim, J., Kim, T.: META-BTS: Bootstrapping precision beyond the limit. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 223–234. ACM Press (Nov 2022). <https://doi.org/10.1145/3548606.3560696>
 4. Bae, Y., Cheon, J.H., Kim, J., Stehlé, D.: Bootstrapping bits with CKKS. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part II. LNCS, vol. 14652, pp. 94–123. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58723-8_4
 5. Bae, Y., Kim, J., Stehlé, D., Suvanto, E.: Bootstrapping small integers with CKKS. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 330–360. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_11
 6. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: Elliptic curves for security. RFC 7748 (2016). <https://doi.org/10.17487/RFC7748>
 7. Boneh, D., Kim, J.: Homomorphic encryption for large integers from nested residue number systems. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part III. LNCS, vol. 16002, pp. 338–370. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01881-6_11
 8. Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 587–617. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_21
 9. Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. LNCS, vol. 13269, pp. 521–541. Springer, Cham (Jun 2022). https://doi.org/10.1007/978-3-031-09234-3_26
 10. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology* **14**(1), 316–338 (2020). <https://doi.org/doi:10.1515/jmc-2019-0026>, <https://doi.org/10.1515/jmc-2019-0026>
 11. Brakerski, Z., Friedman, O., Golan, D., Gurny, A., Mutzari, D., Sheinfeld, O.: REFHE: Fully homomorphic ALU. In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part IV. LNCS, vol. 16545, pp. 183–213. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25327-9_7
 12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012). <https://doi.org/10.1145/2090236.2090262>
 13. Cha, G., Park, D., Lee, J.W.: Improved radix-based approximate homomorphic encryption for large integers via lightweight bootstrapped digit carry. In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part IV. LNCS, vol. 16545, pp. 243–273. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25327-9_9
 14. Cheon, J.H., Cho, W., Kim, J., Stehlé, D.: Homomorphic multiple precision multiplication for CKKS and reduced modulus consumption. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 696–710. ACM Press (Nov 2023). <https://doi.org/10.1145/3576915.3623086>
 15. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I.

- LNCS, vol. 10624, pp. 409–437. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_15
16. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 3–33. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_1
 17. Drucker, N., Moshkovich, G., Pelleg, T., Shaul, H.: BLEACH: Cleaning errors in discrete computations over CKKS. *Journal of Cryptology* **37**(1), 3 (Jan 2024). <https://doi.org/10.1007/s00145-023-09483-1>
 18. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 617–640. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_24
 19. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive, Report 2012/144* (2012), <https://eprint.iacr.org/2012/144>
 20. Gao, M., Zheng, H.: FHE for SIMD arithmetic logic units with amortized $O(1)$ bootstrapping per ciphertext. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology – CRYPTO 2026, Part II*. LNCS, vol. 16801, pp. 299–330. Springer (Aug 2026). https://doi.org/10.1007/978-3-032-35374-0_10
 21. Geelen, R., Vercauteren, F.: Fully homomorphic encryption for cyclotomic prime moduli. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part III. LNCS, vol. 15603, pp. 366–397. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91131-6_13
 22. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press (May / Jun 2009). <https://doi.org/10.1145/1536414.1536440>
 23. Kim, J.: Efficient homomorphic integer computer from CKKS. *IACR TCHES* **2025**(4), 873–898 (2025). <https://doi.org/10.46586/tches.v2025.i4.873-898>
 24. Kim, J.: Faster homomorphic integer computer. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2026**(3), 1128–1151 (Jul 2026). <https://doi.org/10.46586/tches.v2026.i3.1128-1151>, <https://tches.iacr.org/index.php/TCHES/article/view/13141>
 25. Kim, S., Park, M., Kim, J., Kim, T., Min, C.: EvalRound algorithm in CKKS bootstrapping. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 161–187. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22966-4_6
 26. Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 130–160. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22966-4_5
 27. jie Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 IEEE Symposium on Security and Privacy. pp. 1057–1073. IEEE Computer Society Press (May 2021). <https://doi.org/10.1109/SP40001.2021.00043>
 28. National Institute of Standards and Technology: Digital signature standard (DSS). Tech. Rep. FIPS 186-5, National Institute of Standards and Technology (2023). <https://doi.org/10.6028/NIST.FIPS.186-5>
 29. Peikert, C., Zarchy, D., Zyskind, G.: High-precision exact fhe made simple, general, and fast. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology – CRYPTO 2026*. pp. 525–557. Springer Nature Switzerland, Cham (2026)

30. Wang, R., Wen, Y., Li, Z., Lu, X., Wei, B., Liu, K., Wang, K.: Circuit bootstrapping: Faster and smaller. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part II. LNCS, vol. 14652, pp. 342–372. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58723-8_12
31. Zama: TFHE-rs: A Rust implementation of the TFHE scheme. Software (2026), <https://github.com/zama-ai/tfhe-rs>

A Detailed Proofs

A.1 Arithmetic Polynomial Encoding

Proof (Proof of Theorem 1). Every coordinate of the CKKS rounding error $\tilde{a} - \Delta\sigma^{-1}(\mathbf{z})$ has magnitude at most $1/2$. Applying σ gives the corresponding slot error, and restricting to one message block does not increase its infinity norm. Applying ρ^{-1} to this block gives the coefficient error $e_{\text{enc},r}$. By the induced operator norms, $\|e_{\text{enc},r}\|_{\infty} \leq \|\mathbf{V}_{\rho}\|_{\infty} \|\sigma\|_{\infty}$. The stated theorem uses this slightly looser bound with the factor $1/2$ omitted. The same argument applies independently to every packed message.

Proof (Proof of Theorem 2). The sum of the two plaintexts is $\Delta(A + A') + e + e'$. Since evaluation at $X = b$ induces a ring homomorphism to $\mathbb{Z}_M$, there exists $C_+ \in \mathcal{P}_M$ such that $[m]_{F_M} + [m']_{F_M} = [m + m']_{F_M} + tC_+$. Hence, $A + A' = [m + m']_{F_M} + t(I + I' + C_+)$. The stated overflow and approximation-error bounds follow from the triangle inequality.

Proof (Proof of Theorem 3). Expanding AA' gives

$$I_{\times} = C_{\times} + I[m']_{F_M} + I'[m]_{F_M} + tII'.$$

Applying $\|ab\|_{\infty} \leq \mu_M \|a\|_{\infty} \|b\|_{\infty}$ and $\|ta\|_{\infty} \leq b\mu_M \|a\|_{\infty}$ to these terms gives the claimed bound on $\|I_{\times}\|_{\infty}$.

After division by Δ , the approximation error is $e_{\times} = Ae' + A'e + ee'/\Delta$. Moreover, $\|A\|_{\infty} \leq \|[m]_{F_M}\|_{\infty} + b\mu_M \|I\|_{\infty}$, with the analogous bound for A' . Applying the product bound to the three terms of $e_{\times}$ and then the triangle inequality gives the claimed error bound.

A.2 Arithmetic Bootstrapping

Proof (Proof of Theorem 4). After $\text{StC}_{\mathcal{P}_M}^{(t^{-1})}$ and $\text{StC}_{\mathcal{R}}$, the decrypted coefficient polynomial is $\Delta(t^{-1}[m]_{F_M} + I) + e_c$. Since I has integer coefficients, $\text{Truncate}_{\Delta \rightarrow q_0}$ removes its contribution modulo q_0 . Centered reduction may introduce an integer polynomial I' such that the coefficients of $t^{-1}[m]_{F_M} + e_c/\Delta + I'$ lie in the centered interval, which gives $\|I'\|_{\infty} \leq \lfloor \|t^{-1}[m]_{F_M}\|_{\infty} + \|e_c\|_{\infty}/\Delta + 1/2 \rfloor$. By Lemma 1, modulus raising introduces another integer polynomial I'' satisfying $\|I''\|_{\infty} \leq 15$ except with negligible probability. Multiplication by t then gives $[m]_{F_M} + t(I' + I'')$, so $I_{\text{out}} = I' + I''$ and

$$\|I_{\text{out}}\|_{\infty} \leq 15 + \left\lceil \|t^{-1}[m]_{F_M}\|_{\infty} + \frac{\|e_c\|_{\infty}}{\Delta} + \frac{1}{2} \right\rceil.$$

The bound does not depend on the magnitude of the input overflow polynomial I .

Proof (Proof of Theorem 5). Immediately before $\text{Truncate}_{\Delta \rightarrow q_0}$, let the ciphertext decrypt to $\Delta A + e_{\text{tr}}$, where $\|e_{\text{tr}}\|_{\infty} \leq \|e\|_{\infty} + \varepsilon_c$. Since A has integer coefficients, $\text{Truncate}_{\Delta \rightarrow q_0}$ removes the contribution of A modulo q_0 and scales the remaining error by q_0/Δ . The first condition implies that this error has magnitude below $q_0/2$, so no wraparound occurs modulo q_0 .

The operations preceding EvalMod_{q_0} contribute at most ε_m additional error, so its input error is at most $(q_0/\Delta)(\|e\|_{\infty} + \varepsilon_c) + \varepsilon_m$. The second condition places this value inside the admissible range B_{mod} of EvalMod_{q_0} . For the scaled sine approximation used by EvalMod_{q_0} , the deviation from exact reduction for an input error of magnitude x is bounded by $(2\pi^2/3)|x|^3/q_0^2$ [8,20]. After restoring the scale to Δ , this contributes at most $(2\pi^2/3)\Delta((\|e\|_{\infty} + \varepsilon_c)/\Delta + \varepsilon_m/q_0)^3$. Subtracting the recovered error would leave exactly ΔA if the remaining homomorphic operations were exact. Their residual approximation error contributes at most $\|e_A\|_{\infty}$, and the result follows from the triangle inequality.

Proof (Proof of Theorem 6). After modulus raising and $\text{CtS}_{\mathcal{R}}$, the coefficient slots contain $q_0(t^{-1}[m]_{F_M} + I_{\text{out}})$ up to approximation error. Multiplication by t modulo F_M gives $q_0([m]_{F_M} + tI_{\text{out}}) + e_r$. Since $[m]_{F_M} + tI_{\text{out}}$ has integer coefficients and $\|e_r\|_{\infty} < B_{\text{round}}$, EvalRound_{q_0} preserves the corresponding multiples of q_0 while removing the approximation error [25]. Finally, $\text{CtS}_{\mathcal{P}_M}^{(\Delta/q_0)}$ returns to the arithmetic root representation and restores the scale to Δ . Hence, the output represents the same residue m with overflow polynomial I_{out} .

A.3 Arbitrary Modular Arithmetic

Proof (Proof of Lemma ??). Every coefficient of a valid input polynomial has absolute value at most $17b + 15$. Hence, the j -th lower coefficient of $P(X)$ is bounded by $(j+1)(17b+15)^2$. After multiplication by $\text{Rad}_b(-M^{-1} \bmod b^n)$, whose coefficients are at most $b-1$, the j -th coefficient below degree n is bounded by

$$\frac{(j+1)(j+2)}{2}(b-1)(17b+15)^2.$$

Reduction modulo $X^n - X + b$ does not change these contributions to the evaluation at $X = b$ except by multiples of b^n . Therefore,

$$|Q(b)| < \frac{n(n+1)}{2}(17b+15)^2b^n.$$

Since $Q(b) = q + sb^n$ and $0 \leq q < b^n$, we obtain $|s| \leq S$.

B Details of the Arithmetic Polynomial Construction

B.1 Details of the Defining Polynomial Search

We give the details of the search described in Section 3.4. We fix M , b , an even degree d , and a monic reference polynomial G with d distinct nonreal roots on the unit circle. Our concrete searches use $G_0(X) = X^d + 1$ or $G_1(X) = 1 + X + \dots + X^d$.

CVP lattice. Choose $E_0(X) \in \mathbb{Z}[X]$ with $\deg E_0 < d$ and $E_0(b) = M - G(b)$. Every monic degree- d integer polynomial satisfying $F(b) = M$ has the unique form

$$F(X) = G(X) + E_0(X) + (X - b)H(X), \quad H(X) = \sum_{j=0}^{d-2} h_j X^j, \quad (7)$$

with $\mathbf{h} = (h_0, \dots, h_{d-2})^\top \in \mathbb{Z}^{d-1}$.

Let $\omega_0, \dots, \omega_{d-1}$ be the roots of G . Define $A \in \mathbb{C}^{d \times (d-1)}$ and $\mathbf{y} \in \mathbb{C}^d$ by $A_{k,j} = (\omega_k - b)\omega_k^j$ and $y_k = -E_0(\omega_k)$. Let $\tilde{A} \in \mathbb{R}^{2d \times (d-1)}$ and $\tilde{\mathbf{y}} \in \mathbb{R}^{2d}$ denote the corresponding real embeddings. For the candidate indexed by $\mathbf{h}$, Equation 7 gives $\|\tilde{A}\mathbf{h} - \tilde{\mathbf{y}}\|_2^2 = \sum_{k=0}^{d-1} |F(\omega_k)|^2$. Thus, the search metric is exactly the Euclidean distance from the target $\tilde{\mathbf{y}}$ to the lattice $\tilde{A}\mathbb{Z}^{d-1}$. Since the roots ω_k are distinct and $\omega_k \neq b$, the matrix $\tilde{A}$ has full column rank.

Arithmetic-growth bound. For $0 \leq k \leq 2d - 2$, write $X^k \bmod F = \sum_{\ell=0}^{d-1} c_{k,\ell} X^\ell$. Let $n_k = k + 1$ for $k < d$ and $n_k = 2d - 1 - k$ for $k \geq d$, and define

$$\mu(F) = \max_{0 \leq \ell < d} \sum_{k=0}^{2d-2} n_k |c_{k,\ell}|. \quad (8)$$

Lemma 2. *For all $A, B \in \mathcal{P}_{M,\mathbb{R}}$, $\|AB\|_\infty \leq \mu(F) \|A\|_\infty \|B\|_\infty$. If $F(X) = X^d + \sum_{i=0}^{d-1} f_i X^i$, then $\mu(F) \geq (d-1) \|\mathbf{f}\|_\infty$.*

Proof. Write $A = \sum_i a_i X^i$ and $B = \sum_j b_j X^j$. Before reduction modulo F , the coefficient of X^k in AB has magnitude at most $n_k \|A\|_\infty \|B\|_\infty$. Replacing X^k by its degree-less-than- d representative modulo F and applying the triangle inequality gives the first inequality.

For $k = d$, we have $X^d \bmod F = -\sum_{i=0}^{d-1} f_i X^i$. Since $n_d = d - 1$, the $k = d$ term in Equation 8 gives $\mu(F) \geq (d-1)|f_i|$ for every i .

For a selected defining polynomial F_M , we take $\mu_M = \mu(F_M)$ in the arithmetic bounds of Section 3.4.

Additional admissibility bounds. Let $\kappa_{\max} = \lfloor (M-1)/b^d \rfloor$. Equation 1 gives $\|[m]_F\|_\infty \leq b - 1 + \kappa_{\max} \|\mathbf{f}\|_\infty$ for every $0 \leq m < M$. When $M \leq b^d$, this reduces to $\|[m]_F\|_\infty \leq b - 1$.

Algorithm 3 Search for a defining polynomial

Require: M, b , even degree d , reference polynomial G , and prescribed admissibility bounds

Ensure: The defining polynomials satisfying all prescribed bounds

- 1: Choose E_0 with $\deg E_0 < d$ and $E_0(b) = M - G(b)$
 - 2: Construct $\tilde{A}$ and $\tilde{\mathbf{y}}$
 - 3: Determine $R_{\max}$ from Equation 9
 - 4: Enumerate all $\mathbf{h} \in \mathbb{Z}^{d-1}$ satisfying $\|\tilde{A}\mathbf{h} - \tilde{\mathbf{y}}\|_{\infty} \leq R_{\max}$
 - 5: **for** each corresponding polynomial F **do**
 - 6: Retain F only if $\gcd(F, F') = 1$
 - 7: Retain F only if F has no real root
 - 8: Retain F only if $\mu(F) \leq B_{\mu}$
 - 9: Retain F only if all remaining admissibility bounds hold
 - 10: **end for**
 - 11: **return** the retained polynomials
-

Let $\xi_0, \dots, \xi_{d/2-1}$ denote the roots of F in the upper half plane. For the transformations of Section 3.2, we bound $\|\mathbf{U}_{\rho}\|_{\infty \rightarrow \infty}$ and $2\|\mathbf{V}_{\rho}\|_{\infty \rightarrow \infty}$. We also bound $\max_j |\xi_j - b|$ and $1/\min_j |\xi_j - b|$, which control multiplication by $X - b$ and its inverse in the root representation. The roots are isolated in disjoint complex intervals so that these bounds can be certified.

Finite search region. Fix an admissible upper bound B_{μ} on $\mu(F)$. By Lemma 2, every candidate satisfying this bound obeys $\|\mathbf{f}\|_{\infty \rightarrow \infty} \leq B_{\mu}/(d-1)$. Let $\mathbf{g}$ denote the coefficient vector of G below degree d . For the reference polynomials used here, $\|\mathbf{g}\|_{\infty \rightarrow \infty} \leq 1$, so $\|\mathbf{f} - \mathbf{g}\|_{\infty \rightarrow \infty} \leq B_{\mu}/(d-1) + 1$. Since every root ω_k of G lies on the unit circle, $|F(\omega_k)| = |(F - G)(\omega_k)| \leq d(B_{\mu}/(d-1) + 1)$. Consequently, every candidate satisfying the multiplication bound lies inside

$$\|\tilde{A}\mathbf{h} - \tilde{\mathbf{y}}\|_2 \leq R_{\max} := d^{3/2} \left(\frac{B_{\mu}}{d-1} + 1 \right). \quad (9)$$

Because $\tilde{A}$ has full column rank, this bounded region contains only finitely many lattice vectors $\tilde{A}\mathbf{h}$. Hence, every candidate satisfying the prescribed multiplication bound occurs in a finite CVP search region.

Candidate conditions. For every lattice vector in Equation 9, Equation 7 determines a unique defining polynomial F . We retain only polynomials with distinct nonreal roots and satisfying the prescribed bounds on $\mu(F)$, the representative size, the coefficient-to-root transformations, and multiplication by $X - b$ and its inverse.

Distinctness is certified by $\gcd(F, F') = 1$. A real-root count determines whether any real root remains. The numerical bounds associated with the nonreal roots are certified from disjoint root-isolating intervals.

Theorem 7 (Completeness of the search). *Fix M, b , an even degree d , a reference polynomial G , and finite admissibility bounds including $\mu(F) \leq B_{\mu}$.*

Algorithm 3 terminates and examines every monic polynomial $F \in \mathbb{Z}[X]$ of degree d satisfying $F(b) = M$ and all prescribed bounds. If no such polynomial exists, the algorithm returns the empty set.

Proof. Equation 7 gives a unique vector $\mathbf{h} \in \mathbb{Z}^{d-1}$ for every monic degree- d integer polynomial satisfying $F(b) = M$. Lemma 2 implies that every polynomial satisfying $\mu(F) \leq B_\mu$ also satisfies $\|\mathbf{f}\|_\infty \leq B_\mu/(d-1)$. Equation 9 therefore places every admissible polynomial inside the enumerated CVP region.

Since $\tilde{A}$ has full column rank, this bounded region contains finitely many lattice vectors. The enumeration therefore terminates and includes every polynomial that can satisfy the prescribed bounds. The subsequent root and numerical conditions remove exactly the candidates that violate one of those bounds. Thus, every admissible polynomial is retained, and the returned set is empty exactly when no admissible polynomial exists.

Arbitrary moduli. The search depends only on M , b , and d and does not use a factorization of M . It therefore applies without change to composite moduli, including RSA moduli. The completeness statement above concerns the search for fixed (M, b, d) and prescribed admissibility bounds; it does not assert that such a polynomial exists for every parameter choice.

B.2 Constructions for the Example Moduli

We give the root, packing, and coefficient-growth bounds for the defining polynomials used for P-384 and Curve25519. Both constructions use $b = 2$ and $t = X - 2$. For each polynomial, $\gcd(F_M, F'_M) = 1$, and exact real-root counting shows that no root is real. Hence, the roots form distinct nonreal conjugate pairs and the packing of Section 3.2 applies.

For the arithmetic bounds below, we take $\mu_M = \mu(F_M)$ as defined in Equation 8.

P-384 For $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$, we use

$$F_{384}(X) = X^{384} - X^{128} - X^{96} + X^{32} - X^2 + 3.$$

It satisfies $F_{384}(2) = p_{384}$ and has 384 distinct nonreal roots. Hence, one message occupies 192 complex slots, and for $N = 2^{16}$ one ciphertext packs $\lfloor N/384 \rfloor = 170$ messages.

The representative is $[m]_{F_{384}} = \sum_{i=0}^{383} m_i X^i$, so $\|[m]_{F_{384}}\|_\infty \leq 1$. The reductions $X^k \bmod F_{384}$ give $\mu_{384} = 2713$. Multiplication by $t = X - 2$ has coefficient operator norm 5, and therefore $\|ta\|_\infty \leq 5\|a\|_\infty$.

The coefficients of $t^{-1}[m]_{F_{384}}$ satisfy $-3/2 - 2^{-33} < (t^{-1}[m]_{F_{384}})_i < 3/2$. If the normalized approximation error before truncation is smaller than $1/2$, Theorem 4 gives $\|I_{\text{out}}\|_\infty \leq 17$. Consequently, after **OverflowRefresh**, the arithmetic polynomial has coefficient norm at most $1 + 5 \cdot 17 = 86$.

Curve25519 For $p_{25519} = 2^{255} - 19$, we use

$$F_{25519}(X) = \sum_{i=0}^{254} X^i - X^4 - X^2 + 2.$$

It satisfies $F_{25519}(2) = p_{25519}$ and has 254 distinct nonreal roots. Hence, one message occupies 127 complex slots, and for $N = 2^{16}$ one ciphertext packs $\lfloor N/254 \rfloor = 258$ messages.

The representative

$$[m]_{F_{25519}} = \sum_{i=0}^{253} m_i X^i - m_{254} (F_{25519}(X) - X^{254})$$

has degree less than 254 and satisfies $\|[m]_{F_{25519}}\|_{\infty} \leq 3$. The reductions $X^k \bmod F_{25519}$ give $\mu_{25519} = 2508$. Multiplication by $t = X - 2$ again has coefficient operator norm 5.

The coefficients of $t^{-1}[m]_{F_{25519}}$ lie between -1 and $3/2$. Under the same normalized-error condition, Theorem 4 gives $\|I_{\text{out}}\|_{\infty} \leq 17$. Consequently, after **OverflowRefresh**, the arithmetic polynomial has coefficient norm at most $3 + 5 \cdot 17 = 88$.

For both moduli, evaluation at $X = 2$ maps arithmetic modulo the defining polynomial to arithmetic modulo M . Thus, **OverflowRefresh**, **ErrorRefresh**, and **ArithBootstrap** apply with the same algebraic structure, with only the polynomial-dependent transformations and bounds changing.

C Details of Representation Conversion

C.1 General Arithmetic-to-Digit Algorithm

We give the full homomorphic algorithm corresponding to Section 5.2. We use the notation a_j , $f_{j,\lambda}$, c_j , b_λ , s , h , and ℓ from the main text and assume $\|I'\|_{\infty} \leq K_I$.

The candidate set for $\lambda = \kappa + I'_{d-1}$ is $\Lambda = \{-K_I, \dots, \lfloor (M-1)/b^d \rfloor + K_I\}$. The number of radix- b positions needed to represent κ is $h_\kappa = \lceil \log_b(\lfloor (M-1)/b^d \rfloor + 1) \rceil$. For $0 \leq j < \ell$, let $B_j = B$ for $j < \ell - 1$ and $B_{\ell-1} = b^h$. We take the remainder representatives to satisfy $0 \leq r_j < B_j$. For the correct value of λ , every carry lies in

$$\left[\left\lceil \frac{-K_I - \max_{\lambda \in \Lambda} |\lambda \alpha_j| - (B_j - 1)}{B_j} \right\rceil, \left\lfloor \frac{B_j - 1 + K_I + \max_{\lambda \in \Lambda} |\lambda \alpha_j|}{B_j} \right\rfloor \right]. \quad (10)$$

We let $\mathcal{C}$ contain the union of these intervals together with 0. If $f_{j,\lambda}(c) \notin \mathcal{C}$, its outgoing indicators are defined to be zero. Equation 10 ensures that this convention does not remove the carry sequence associated with the correct candidate.

In Algorithm 4, the index j denotes the relative slot position within one occupied interval. The indicator vectors are public plaintext masks repeated

independently in each occupied interval. All assignments at one doubling distance use the ciphertext values at the beginning of that iteration.

The second functional bootstrap returns $\text{ct}_{\lambda,c,c'}^f[j] = \mathbf{1}[f_{j,\lambda}(c) = c']$ for $\lambda \in \Lambda$, $c, c' \in \mathcal{C}$, and $0 \leq j < \ell$. At position ℓ , for $\lambda \in \Lambda$, $c \in \mathcal{C}$, and $0 \leq g < h_\kappa$, it returns

$$\begin{aligned} \text{ct}_{\lambda,c}^{\text{chk}}[\ell] &= \mathbf{1} \left[|c - q_{\ell-1}| \leq K_I, 0 \leq \lambda - c + q_{\ell-1} \leq \left\lfloor \frac{M-1}{b^d} \right\rfloor \right], \\ \text{ct}_{\lambda,c,g}^\kappa[\ell] &= \text{ct}_{\lambda,c}^{\text{chk}}[\ell] \left(\left\lfloor \frac{\lambda - c + q_{\ell-1}}{b^g} \right\rfloor \bmod b \right). \end{aligned}$$

All other slots of $\text{ct}_{\lambda,c}^{\text{chk}}$ and $\text{ct}_{\lambda,c,g}^\kappa$ are zero.

After the carry composition, $\text{ct}_\lambda^c[j]$ contains the candidate carry $c_{j+1,\lambda}$. The prefix products form the exclusive selectors $b_{\lambda_i} \prod_{k < i} (1 - b_{\lambda_k})$ of Equation 6. The subsequent rotations replicate each selector to the radix positions used in the final reconstruction.

Theorem 8 (Correctness of general arithmetic-to-digit conversion).

Algorithm 4 returns the canonical radix- B digits of the encrypted residue modulo M .

Proof. The input to $\text{FBS}_B^{(2)}$ contains $a_0 = r_0$ at position 0, $a_j = r_j + q_{j-1}$ at positions $1 \leq j < \ell$, and $q_{\ell-1}$ at position ℓ . Each carry-composition iteration implements Equation 4. The plaintext masks retain positions already determined from the initial carry and remove terms crossing an interval boundary. After $\lceil \log_2 \ell \rceil$ iterations, $\text{ct}_{\lambda,0,c}^f[j] = \mathbf{1}[c_{j+1,\lambda} = c]$. Hence, $\text{ct}_\lambda^c[j] = c_{j+1,\lambda}$, and the final-carry indicator is aligned with its validity condition at position ℓ .

Equation 6 selects the smallest valid candidate. By Equation 5, this candidate is the actual value $\lambda = \kappa + I'_{d-1}$. Therefore, $\text{ct}_{c^+}[j] = c_{j+1}$ and $\text{ct}_\lambda[j] = \lambda$ at every radix position.

For $j < \ell - 1$, the reconstruction gives $a_j - c_j + Bc_{j+1} + \lambda\alpha_j$, while the final radix position gives $a_{\ell-1} - c_{\ell-1} + b^h c_\ell + \lambda\alpha_{\ell-1}$. These are exactly the radix- B digits of m_0 . The last loop adds the selected radix- b digits of κ at their weights b^{d+g} . Thus, the returned ciphertext contains the radix- B digits of $m_0 + \kappa b^d = m$.

Algorithm 4 Arithmetic-to-digit conversion over a general modulus

Require: Arithmetic ciphertext ct_A

Ensure: $\text{ct}_y[j]$ contains the j -th radix- B digit of m

```

1:  $(\text{ct}_r, \text{ct}_q) \leftarrow \text{FBS}_B^{(1)}(\text{ct}_A)$ 
2:  $\text{ct}_a \leftarrow \text{PMult}(\text{ct}_r, (\mathbf{1}[0 \leq j < \ell])_j) + \text{PMult}(\text{Rot}_{-1}(\text{ct}_q), (\mathbf{1}[1 \leq j \leq \ell])_j)$ 
3:  $(\{\text{ct}_{\lambda,c,c'}^f\}, \{\text{ct}_{\lambda,c}^{\text{chk}}\}, \{\text{ct}_{\lambda,c,g}^\kappa\}) \leftarrow \text{FBS}_B^{(2)}(\text{ct}_a)$ 
4: for  $\delta = 1, 2, 4, \dots < \ell$  do
5:   Simultaneously for all  $\lambda \in \Lambda$  and  $c, c' \in \mathcal{C}$ :
6:      $\text{ct}_{\lambda,c,c'}^f \leftarrow \text{PMult}(\text{ct}_{\lambda,c,c'}^f, (\mathbf{1}[0 \leq j < \delta])_j)$ 
        $+ \sum_{u \in \mathcal{C}} \text{CMult}(\text{PMult}(\text{Rot}_{-\delta}(\text{ct}_{\lambda,c,u}^f), (\mathbf{1}[\delta \leq j < \ell])_j), \text{ct}_{\lambda,u,c'}^f)$ 
7: end for
8: for  $\lambda \in \Lambda$  do
9:    $\text{ct}_\lambda^c \leftarrow \sum_{c \in \mathcal{C}} \text{PMult}(\text{ct}_{\lambda,0,c}^f, (c \mathbf{1}[0 \leq j < \ell])_j)$ 
10:  for  $c \in \mathcal{C}$  do
11:     $\text{ct}_{\lambda,c}^{\text{end}} \leftarrow \text{PMult}(\text{Rot}_{-1}(\text{ct}_{\lambda,0,c}^f), (\mathbf{1}[j = \ell])_j)$ 
12:  end for
13:   $\text{ct}_\lambda^b \leftarrow \sum_{c \in \mathcal{C}} \text{CMult}(\text{ct}_{\lambda,c}^{\text{end}}, \text{ct}_{\lambda,c}^{\text{chk}})$ 
14:  for  $0 \leq g < h_\kappa$  do
15:     $\text{ct}_{\lambda,g}^\kappa \leftarrow \sum_{c \in \mathcal{C}} \text{CMult}(\text{ct}_{\lambda,c}^{\text{end}}, \text{ct}_{\lambda,c,g}^\kappa)$ 
16:  end for
17: end for
18: Order  $\Lambda$  as  $\lambda_1 < \dots < \lambda_{|\Lambda|}$ 
19: for  $1 \leq i \leq |\Lambda|$  do
20:    $\text{ct}_i^{\text{pre}} \leftarrow (\mathbf{1}[j = \ell])_j - \text{ct}_{\lambda_i}^b$ 
21: end for
22: for  $\delta = 1, 2, 4, \dots < |\Lambda|$  do
23:   Simultaneously for all  $\delta < i \leq |\Lambda|$ :
24:     $\text{ct}_i^{\text{pre}} \leftarrow \text{CMult}(\text{ct}_{i-\delta}^{\text{pre}}, \text{ct}_i^{\text{pre}})$ 
25: end for
26: for  $2 \leq i \leq |\Lambda|$  do
27:    $\text{ct}_{\lambda_i}^b \leftarrow \text{CMult}(\text{ct}_{\lambda_i}^b, \text{ct}_{i-1}^{\text{pre}})$ 
28: end for
29: for  $1 \leq i \leq |\Lambda|$  do
30:    $\text{ct}_{\lambda_i}^b \leftarrow \sum_{0 \leq u < \max\{\ell+1, \lceil \log_B M \rceil\}} \text{PMult}(\text{Rot}_{\ell-u}(\text{ct}_{\lambda_i}^b), (\mathbf{1}[j = u])_j)$ 
31: end for
32:  $\text{ct}_{c+} \leftarrow \sum_{i=1}^{|\Lambda|} \text{CMult}(\text{ct}_{\lambda_i}^b, \text{ct}_{\lambda_i}^c)$ 
33:  $\text{ct}_\lambda \leftarrow \sum_{i=1}^{|\Lambda|} \text{PMult}(\text{ct}_{\lambda_i}^b, (\lambda_i \mathbf{1}[0 \leq j < \ell])_j)$ 
34:  $\text{ct}_y \leftarrow \text{PMult}(\text{ct}_a, (\mathbf{1}[0 \leq j < \ell])_j)$ 
        $- \text{PMult}(\text{Rot}_{-1}(\text{ct}_{c+}), (\mathbf{1}[1 \leq j < \ell])_j)$ 
        $+ \text{PMult}(\text{ct}_{c+}, (B \mathbf{1}[0 \leq j < \ell-1] + b^h \mathbf{1}[j = \ell-1])_j)$ 
        $+ \text{PMult}(\text{ct}_\lambda, (\alpha_j \mathbf{1}[0 \leq j < \ell])_j)$ 
35: for  $0 \leq g < h_\kappa$  do
36:    $\text{ct}_{\kappa,g} \leftarrow \sum_{i=1}^{|\Lambda|} \text{CMult}(\text{ct}_{\lambda_i}^b, \text{ct}_{\lambda_i,g}^\kappa)$ 
37:    $\text{ct}_y \leftarrow \text{ct}_y + \text{PMult}(\text{Rot}_{\ell - \lfloor (d+g)/s \rfloor}(\text{ct}_{\kappa,g}), (b^{(d+g) \bmod s} \mathbf{1}[j = \lfloor (d+g)/s \rfloor])_j)$ 
38: end for
39: return  $\text{ct}_y$ 

```

C.2 Cost of General Arithmetic-to-Digit Conversion

The conversion uses exactly two functional bootstrap calls. The second bootstrap returns $|A|(|C|^2 + |C|(1 + h_\kappa))$ ciphertexts corresponding to the carry indicators, the candidate checks, and the radix- b digits of each candidate κ .

Each of the $\lceil \log_2 \ell \rceil$ carry-composition levels uses $|A||C|^2$ rotations, $2|A||C|^2$ ciphertext–plaintext multiplications, and $|A||C|^3$ ciphertext–ciphertext multiplications. All products within one composition level are independent, so the carry computation has multiplicative depth $\lceil \log_2 \ell \rceil$.

Combining the final-carry indicators with the candidate checks and the digits of κ adds one multiplicative level. The prefix products over A require $\lceil \log_2 |A| \rceil$ levels, and forming and applying the exclusive selectors adds two further levels. All remaining rotations and digit reconstruction are linear, so the multiplicative depth following the second bootstrap is at most $\lceil \log_2 \ell \rceil + \lceil \log_2 |A| \rceil + 3$. The prefix products use $\sum_{\delta \in \{1, 2, 4, \dots\}, \delta < |A|} (|A| - \delta)$ ciphertext–ciphertext multiplications, and forming the exclusive selectors uses another $|A| - 1$. The final selection uses $|A|(1 + h_\kappa)$ ciphertext–ciphertext multiplications. The same operations act simultaneously on all occupied intervals in a SIMD batch.

C.3 Choice of the Radix Base

We specialize to $M = 2^n$, with $B = 2^s$ and $\ell = \lceil n/s \rceil$. The bound $|I'_i| \leq 16$ determines the number of carry states required by the radix reduction. We use $B = 32$ in our implementation because it is the smallest power-of-two radix for which two carry states suffice. Other radix choices follow the same conversion framework.

Radices at least 32 Suppose $B \geq 32$ and choose representatives in the centered interval $-B/2 \leq r_j < B/2$. Since $0 \leq d_j < B$ and $|I'_{sj-1}| \leq 16 \leq B/2$, the relation $r_j = d_j + I'_{sj-1} - Bc_{j+1}$ requires only $c_{j+1} \in \{0, 1\}$. Adding the quotient from the preceding radix position cancels the boundary overflow coefficient and gives $a_j = d_j + c_j - Bc_{j+1}$.

Increasing B decreases the number ℓ of radix positions and can reduce the depth of the carry composition. At the same time, it increases the number of bit functions evaluated at each radix position and the size of the lookup implemented by $\text{FBS}_B^{(2)}$.

Radices below 32 For $B < 32$, two carry states no longer suffice in general. Using representatives $0 \leq r_j < B$, the possible carry values for the power-of-two radices $B \leq 16$ are $\mathcal{C} = \{-16/B, -16/B + 1, \dots, 16/B\}$. Thus, $B = 16, 8$, and 4 require 3, 5, and 9 carry states, respectively. The same cancellation gives $a_j = d_j + c_j - Bc_{j+1}$. For a fixed incoming carry c_j , the next carry and digit are $c_{j+1} = \lceil (c_j - a_j)/B \rceil$ and $d_j = (a_j - c_j) \bmod B$. The second functional bootstrap evaluates these functions for every carry state, and the resulting carry maps are composed with the same logarithmic-depth procedure as in Section 5.2.

For example, $B = 16$ gives $\mathcal{C} = \{-1, 0, 1\}$. For each possible incoming carry, $\text{FBS}_{16}^{(2)}$ evaluates the next carry and the four output bits of the corresponding radix digit.

Minimum number of carry states The value $d_j + I'_{sj-1}$ ranges over the $B + 32$ consecutive integers from -16 to $B + 15$. For a fixed carry value, an interval of B remainder representatives covers at most B of these integers, so any radix reduction requires at least $\lceil (B + 32)/B \rceil$ carry states.

For $B \geq 32$, this lower bound is two and is attained by centered reduction. For $B \leq 16$, the bound is $1 + 32/B$ and is attained by the representatives above. Hence, $B = 32$ is the smallest power-of-two radix with a binary carry. Smaller radices reduce the lookup radix at the cost of more carry states, while larger radices retain a binary carry but require larger lookup functions.