

Fully Homomorphic Encryption for Small Batches of Complex Numbers

Jules Dumezy

University of Paris-Saclay, CEA-List, France
`jules.dumezy@cea.fr`

Abstract. We propose a new approximate homomorphic encryption scheme to process small batches of complex numbers. Leveled operations are based on a new *shaped rounding*, whose error is a multiple of a fixed public ring element g , small on the data slots and larger on the others. The unused slots form a guard band that is computed on but never decoded, and the polynomial scaling factor $g^{-1}\Delta$ generalizes the CKKS scaling factor, with $g = 1$ recovering CKKS. Guard-domain bounds determine the scaling factor needed for stability, while a norm bound limits the precision gain for a given guard amplification. A replicated layout uses a unit from a small cyclotomic subring to preserve shaping under admissible rotations and traces. We give a native bootstrapping algorithm whose forward transform reads only data slots, and conversions to and from CKKS with shared keys. Security reduces to Ring-LWE with the same parameters as CKKS. In poulpy, at ring degree 2^{15} and parameters targeting 128-bit security, bootstrapping a ciphertext holding 16 or 64 values restores 21 levels and retains about 22.8 bits of minimum precision after 21 sequential products by fresh operands, while sparse CKKS supports only four levels at that degree and comparable precision. For these batch sizes, bootstrapping to restore 16 levels is about 3.6 times faster on one thread than sparse CKKS, which requires ring degree 2^{16} .

Keywords: Approximate Homomorphic Encryption · Bootstrapping · Ring-LWE

Table of Contents

1	Introduction	3
2	Preliminaries	6
2.1	Notation and Cyclotomic Rings	6
2.2	Ring-LWE Encryption	8
2.3	Homomorphic Encryption	8
3	The Proposed Scheme	9
3.1	Shaped Rounding	9
3.2	Layouts	10
3.3	Encoding and Encryption	11
3.4	Homomorphic Operations	12
3.5	Security Analysis	14
4	Scheme Analysis	15
4.1	Data Error, Guard Error and the Scaling Factor Law	15
4.2	Correctness of General Circuits	18
4.3	Practical Scaling Factor Selection	20
4.4	Limits of Shaping	21
5	Bootstrapping Our Scheme	22
5.1	Fiber-Aware Linear Transformations	23
5.2	Precision During the Linear Transforms	25
5.3	EvalMod and Its Constants	26
5.4	Correctness and Modulus Budget	28
6	Instantiation and Implementation	29
6.1	Choosing the Shaping Unit	29
6.2	Encoding and Decoding in the Subring	30
6.3	The Shaped-Rounding Pass	31
6.4	Representation and Evaluation Keys	32
7	Experimental Results	33
7.1	Experimental Setup	33
7.2	Bootstrapping Latency and Available Depth	34
7.3	Leveled Precision and Stability	34
7.4	Comparison with PaCo	37
A	Conversions to and from CKKS	41
A.1	From CKKS to the Replicated Layout	42
A.2	From the Replicated Layout to CKKS	43
B	IND-CPA ^D Security	44
C	Supplementary Guard Analysis	44
C.1	Realified Polynomial Evaluations	44
C.2	Guard-Escape Estimate	45

1 Introduction

Fully Homomorphic Encryption (FHE), introduced by Gentry in 2009 [Gen09], allows arbitrary computations on encrypted data without access to the secret key. The main schemes fall into three families. BFV [Bra12,FV12], its generalization GBFV [GV25], and BGV [BGV12] support exact arithmetic on large batches of integers modulo a plaintext modulus or elements of finite fields. They use single-instruction multiple-data (SIMD) packing to apply the same operation to many values in one ciphertext, amortizing its cost over the batch. FHEW/TFHE [DM15,CGGI20] operates on individual bits or small integers, using functional bootstrapping to combine function evaluation with noise control. CKKS [CKKS17] supports SIMD approximate arithmetic on large batches of real or complex numbers, and recent work also enables efficient computation on integers through dedicated functional bootstrapping [BKSS24,AKP25].

Suppose, however, that a computation involves only a small batch of real numbers or complex numbers. TFHE offers a natural starting point for processing a few values, but its native operations act on bits or small integers, practically up to 4-5 bits. Arithmetic on real or complex numbers must be implemented through a discrete representation, for example by decomposing fixed-point values into small digits. This involves several bootstrapping and is expensive (e.g. [TBC⁺25]).

CKKS provides the required arithmetic, but most of its packing capacity is then unnecessary. The ring dimension is constrained by security and by the ciphertext modulus required for the computation and the desired precision, so it cannot in general be reduced to match the number of values. Using fewer slots therefore leaves much of the cost of each operation unchanged.

We therefore ask whether we can design a scheme that trades unused slots for either of the following benefits:

- Greater precision in the useful slots with the same scaling factor.
- A lower scaling factor at the same precision, reducing modulus consumption.

Our Contribution. We propose a new FHE scheme for small batches of real or complex numbers, based on a new encoding and two new operations: *shaped rescaling* and *shaped keyswitching*. It trades packing capacity for greater precision with a fixed scaling factor, or a lower scaling factor at a fixed precision, reducing the modulus consumption. We establish correctness and error bounds for arithmetic circuits, quantify the limits of this tradeoff, and prove security under the same Ring-LWE assumption as CKKS for a given set of parameters. The scheme retains the CKKS ciphertext and key formats and includes ordinary CKKS as a special case, thus generalizing it.

We give a bootstrapping algorithm for the proposed scheme and conversions to and from CKKS. Each conversion uses one linear transformation, with both schemes sharing the secret and evaluation keys. This allows a CKKS computation to use the proposed scheme for a subcircuit on a small batch and then return to CKKS, and conversely. Moreover, our scheme is compatible with most CKKS

optimizations, e.g. conjugate-invariant encoding [KS19] or discrete CKKS with functional bootstrapping [BKSS24,AKP25].

At ring degree 2^{15} and scaling factor 2^{20} , a tested chain of 16 multiplications by fresh encrypted operands retains 44.7 bits of precision, compared with 5.1 bits for CKKS with the same scaling factor.

We compare our bootstrapping with sparse CKKS [CHK⁺18] in the poulpy library [BDM⁺26] over batches of 16 to 1024 values and 1 to 21 multiplication levels available after bootstrapping. For batches of 16 and 64 values with parameters targeting 128-bit security and similar precision, our bootstrapping restores 21 levels of 20 bits each at ring degree 2^{15} , while the baseline restores only four 36-bit levels at that degree. In our main measurements, the bootstrapped ciphertexts keep 22.8 bits of minimum precision after 21 products by fresh operands, and none of 200 additional bootstrapping runs over five key sets falls below 21.8 bits. When both bootstrapping algorithms are configured to restore 16 levels, our scheme uses degree 2^{15} while the baseline requires degree 2^{16} . Our bootstrapping is then respectively 3.60 and 3.56 times faster on one thread, and 2.39 and 2.82 times faster using all 18 cores of a MacBook equipped with an Apple M5 Pro.

We also compare with PaCo [CS25] under the same evaluation protocol, at comparable precision. PaCo matches our all-core latency for 16 values and up to four subsequent levels, and is 1.3 to 1.5 times slower on one thread. It restores only four to six levels at ring degree 2^{15} , its cost grows linearly beyond 64 values, and it relies on a structured-secret assumption.

Related Work. Sparse packing in CKKS [CHK⁺18] repeats a short vector across the slots to reduce the cost of bootstrapping transformations, which Zheng [Zhe26] further improves by exploiting the repetition pattern. SPRU, by Coron and Köstler [CK25], and PaCo, by Coron and Seuré [CS25], reduce bootstrapping latency by evaluating the decryption equation through complex roots of unity. These approaches retain ordinary CKKS encoding and operations, whereas our construction reduces the error introduced on useful slots by the leveled operations themselves.

Guimarães et al. [GPvL23,GP25] and BatchBoot [LWZ⁺26] extend bootstrapping for individual encrypted integers to batches, with BatchBoot adapting its packing to the batch size. Their batching amortizes discrete operations and does not provide native leveled arithmetic on real or complex values.

GBFV [GV25], extending the polynomial plaintext moduli of Chen et al. [CLPX18], provides a precedent for trading packing capacity for lower noise growth. Its construction changes the finite plaintext ring by replacing the integer plaintext modulus with a polynomial. CKKS instead recovers approximate values from the canonical embedding, without reduction modulo a plaintext ideal, so this substitution does not apply. Replacing the scaling factor used for encoding by a polynomial is not sufficient either, since rescaling must still produce ciphertexts with integer coefficients and controlled errors on the useful slots.

Peikert, Zarchy and Zyskind [PZZ26] represent exact plaintexts modulo a large integer as digit polynomials. Error rates and growth scale with a root of

the plaintext modulus whose order is the number of digits, at the cost of larger ciphertexts. Our construction targets approximate arithmetic instead, and trades packing capacity rather than ciphertext size for precision, keeping the CKKS ciphertext and key formats.

Technical Overview. The useful freedom is that only the data slots need precision. The remaining slots, which we call *guard slots*, only need to stay bounded throughout the computation to ensure correct decryption. Figure 1 shows how these roles are arranged in our replicated layout. One class out of every R slots carries data, and a batch of k values repeats every kR slots. We choose a public polynomial g whose slot magnitudes repeat every R slots, with a small value on the data class and larger values allowed on the guard classes. Multiplying rounding errors by g then gives the same precision guarantee to every copy of a data value, while the guard classes absorb the larger errors.

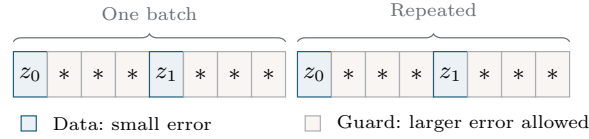


Fig. 1. A replicated batch with $k = 2$ and $R = 4$. Shaping suppresses error on the data slots and allows larger error on the intervening guard slots ($*$), whose values need only stay bounded. Copies of the same intended data may have different errors.

The difficulty is to do this while keeping integer ciphertext coefficients. For a component $a \in \mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$, with N a power of two, we replace rounding after division by an integer r with

$$h = [(g^{-1} \bmod r)a]_r, \quad \text{Round}_{g,r}(a) = \frac{a - gh}{r},$$

where brackets denote centered coefficientwise reduction and g is invertible modulo r . The congruence $gh \equiv a \pmod{r}$ makes the quotient integral, while its error $-gh/r$ has the desired factor g , and h/r has coefficients bounded by $1/2$. We apply this rule to rescaling, keyswitching and encoding, and sample encryption errors as ge . Writing a phase as $\mu = \Delta m + ge$ then gives $g^{-1}\mu = (g^{-1}\Delta)m + e$, which explains the polynomial scaling factor $g^{-1}\Delta$.

We obtain the repeated profile by choosing g in the subring generated by $X^{N/(2R)}$. Rotations by multiples of R slots then fix g , so rotating the data preserves its error suppression. We use units, whose inverses also have integer coefficients, to allow every rescaling factor. Their slot magnitudes have product one, which limits the data precision gain for a given guard amplification.

The amplified guard errors must remain small enough to prevent wrap-around modulo the ciphertext modulus. For instance, squaring sends a guard bound b to $b^2 + c$, where c bounds the injected error, so a bound of $1/2$ is preserved when

$c \leq 1/4$. We choose g and the scaling factor jointly to meet the data precision and guard stability requirements, propagating guard bounds through the circuit and resetting the guards when necessary.

Bootstrapping must gather the useful values without importing the larger guard errors. We introduce fiber-aware linear transforms to exploit a trace onto the subring of slot period k , so each needs only k diagonals when $R \mid k$, with smaller batches padded to R values. The forward map is designed to recover the transformed data after this projection, using only data slots, while the reverse map writes back to the data class with zero on the guards. We control the errors from quantizing and evaluating these maps with a larger intermediate scaling factor, and use shaped arithmetic for the modular-reduction step. The output restores both data precision and bounded guards, while the same layout transformations also provide conversions to and from CKKS.

2 Preliminaries

2.1 Notation and Cyclotomic Rings

For an integer $q \geq 2$, we write $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ and denote by $[a]_q$ the representative of a modulo q in $[-q/2, q/2)$. For $x \in \mathbb{R}$, we write $\lfloor x \rceil$ for rounding to the nearest integer, with ties rounded upward. Both operations apply coefficientwise to polynomials. For $z \in \mathbb{C}$, we denote its complex conjugate by $\bar{z}$, and extend this notation componentwise to vectors. For a vector $\mathbf{v} = (v_i)_i$, we write $\|\mathbf{v}\|_\infty = \max_i |v_i|$. We identify a polynomial with its coefficient vector when applying this norm. All logarithms are in base two.

Ring and Slots. Let $N \geq 4$ be a power of two, and write $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_\mathbb{R} = \mathbb{R}[X]/(X^N + 1)$. We represent elements of these rings by polynomials of degree less than N , with integer and real coefficients respectively, and reduce products modulo $X^N + 1$. We also write $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ for the corresponding ring with coefficients in $\mathbb{Z}_q$.

The slots of $a \in \mathcal{R}_\mathbb{R}$ are its evaluations at one root from each complex conjugate pair of roots of $X^N + 1$. We use the usual CKKS ordering, with $\zeta = e^{i\pi/N}$ and slot $t \in \mathbb{Z}_{N/2}$ given by $\sigma_t(a) = a(\zeta^{5^t})$. Since a has real coefficients, its evaluations at the remaining roots are the complex conjugates of these slot values. The canonical embedding $\sigma: \mathcal{R}_\mathbb{R} \rightarrow \mathbb{C}^{N/2}$ maps a to its slot vector $\mathbf{z} = (\sigma_t(a))_{t \in \mathbb{Z}_{N/2}}$. It is a bijection that takes polynomial addition and multiplication to componentwise addition and multiplication, so every slot vector $\mathbf{z} \in \mathbb{C}^{N/2}$ corresponds to a unique polynomial $\sigma^{-1}(\mathbf{z}) \in \mathcal{R}_\mathbb{R}$.

We write $\|a\|_{\text{can}} = \|\sigma(a)\|_\infty$ for the largest absolute slot value, called the canonical norm. For $a = \sum_{i=0}^{N-1} a_i X^i$, the inverse evaluation formula $a_i = N^{-1} \sum_{j=0}^{N-1} a(\zeta^{2j+1}) \zeta^{-i(2j+1)}$ gives $\|a\|_\infty \leq \|a\|_{\text{can}}$ by the triangle inequality, so a bound on all slot values also bounds all coefficients.

Automorphisms, Subrings and Traces. For an odd integer u taken modulo $2N$, the substitution $\tau_u: a(X) \mapsto a(X^u) \bmod (X^N + 1)$ is an automorphism of $\mathcal{R}$ and $\mathcal{R}_{\mathbb{R}}$. In the slot ordering above, τ_{5^j} rotates the slots by j , while τ_{-1} conjugates each slot, since $\sigma_t(\tau_{5^j}(a)) = \sigma_{t+j}(a)$ and $\sigma_t(\tau_{-1}(a)) = \sigma_t(\overline{a})$. Slot indices are taken modulo $N/2$. Every odd residue modulo $2N$ is of the form $\pm 5^j$, so each of these automorphisms is a rotation, possibly followed by conjugation.

Let $K = \mathbb{Q}[X]/(X^N + 1)$ be the field containing $\mathcal{R}$. For a divisor $d \leq N/2$ of N , the subfield $K_d = \mathbb{Q}(X^d)$ consists of the polynomials whose nonzero coefficients occur only at multiples of d . It has degree N/d over $\mathbb{Q}$, and its integer elements form the subring $\mathbb{Z}[X^d] \subseteq \mathcal{R}$. Writing $n = N/(2d)$, the automorphisms that fix X^d are $\tau_{5^{ni}}$ for $0 \leq i < d$, and the trace from K to K_d is their sum, $\text{Tr}_{K/K_d}(a) = \sum_{i=0}^{d-1} \tau_{5^{ni}}(a)$. On slots, the normalized trace therefore averages the d values whose indices agree modulo n , so

$$\sigma_t(d^{-1} \text{Tr}_{K/K_d}(a)) = d^{-1} \sum_{i=0}^{d-1} \sigma_{t+ni}(a).$$

On coefficients, the normalized trace simply discards the terms whose exponents are not divisible by d , giving $d^{-1} \text{Tr}_{K/K_d}(a) = \sum_{j=0}^{N/d-1} a_{dj} X^{dj}$. Indeed, the automorphisms fixing X^d are the τ_u with $u = 1 + 2Nm/d$ for $0 \leq m < d$, and such a τ_u sends X^ℓ to $X^\ell \omega^{m\ell}$, where $\omega = X^{2N/d}$ satisfies $\omega^d = 1$. The sum $\sum_{m=0}^{d-1} \omega^{m\ell}$ equals d when d divides ℓ , and vanishes otherwise, since $1 - \omega^\ell$ is then invertible in K and annihilates it. Thus the normalized trace is a projection onto K_d that also preserves integer coefficients. The same formulas hold in $\mathcal{R}_{\mathbb{R}}$, where the normalized trace projects onto the real span $\mathbb{R}[X^d]$ of the powers of X^d . In particular, $a \in \mathcal{R}_{\mathbb{R}}$ belongs to $\mathbb{R}[X^d]$ if and only if its slot vector has period n , meaning $\sigma_{t+n}(a) = \sigma_t(a)$ for every t . One direction follows because the rotations in the trace fix $\mathbb{R}[X^d]$, and the other because averaging a vector of period n leaves it unchanged, so that a equals its normalized trace.

Invertibility and Units. Invertibility in $\mathcal{R}_q$ can be characterized by an integer associated with each polynomial, its algebraic norm. For $a \in \mathcal{R}$, it is defined as the product of its evaluations at all roots of $X^N + 1$, given by

$$\text{Norm}(a) = \prod_{j=0}^{N-1} a(\zeta^{2j+1}) = \prod_{t=0}^{N/2-1} |\sigma_t(a)|^2.$$

For nonzero a , this product is a positive integer, as the determinant interpretation below shows.

Lemma 1 (Invertibility modulo an integer). *For every integer $q \geq 2$, an element $a \in \mathcal{R}$ is invertible in $\mathcal{R}_q$ if and only if $\gcd(\text{Norm}(a), q) = 1$.*

Proof. Multiplication by a is represented in the coefficient basis $1, X, \dots, X^{N-1}$ by an integer matrix whose determinant is $\text{Norm}(a)$, since evaluation at the roots diagonalizes this map. Modulo q , this matrix is invertible if and only if its

determinant is coprime to q . If a is invertible in $\mathcal{R}_q$, multiplication by its inverse gives the inverse matrix. Conversely, an inverse matrix gives a solution $b \in \mathcal{R}_q$ to $ab = 1$ by applying it to the coefficient vector of 1. $\square$

A unit is an element $a \in \mathcal{R}$ whose multiplicative inverse also belongs to $\mathcal{R}$, so it is invertible modulo every integer $q \geq 2$. The same determinant argument shows that a is a unit exactly when $\text{Norm}(a) = 1$, because an integer matrix has an integer inverse if and only if its determinant is ± 1 , and the algebraic norm here is nonnegative. In particular, the slot magnitudes of a unit satisfy $\prod_{t=0}^{N/2-1} |\sigma_t(a)| = 1$.

For a power-of-two modulus $q = 2^b$ with $b \geq 1$, invertibility has the simpler criterion that $a(1)$ be odd. Indeed, over $\mathbb{Z}_2$ we have $X^N + 1 = (X + 1)^N$, so a is invertible modulo 2 exactly when $a(1) \neq 0 \pmod{2}$. By Lemma 1, invertibility modulo 2 is equivalent to an odd norm, which is also the condition for invertibility modulo 2^b .

2.2 Ring-LWE Encryption

Fix an integer modulus $q \geq 2$, a secret-key distribution χ_{sk} on $\mathcal{R}$, and an error distribution χ_e on $\mathcal{R}$ with small coefficients. The secret key s is sampled from χ_{sk} , and a ciphertext is a pair $\mathbf{c} = (c_0, c_1) \in \mathcal{R}_q^2$. The private-key encryption and decryption primitives are defined as follows.

- $\text{Enc}_q(s, m)$, for $m \in \mathcal{R}$, samples a uniformly in $\mathcal{R}_q$ and an independent error e from χ_e , then returns $\mathbf{c} = (-as + m + e, a) \pmod{q}$.
- $\text{Dec}_q(s, \mathbf{c})$ returns the phase $\mu = [c_0 + c_1 s]_q \in \mathcal{R}$.

For $\mathbf{c} = \text{Enc}_q(s, m)$, decryption gives $\mu = [m + e]_q$. If $\|m + e\|_\infty < q/2$, centered reduction leaves the polynomial unchanged and $\mu = m + e$ in $\mathcal{R}$. The stronger condition $\|m + e\|_{\text{can}} < q/2$ is also sufficient, since the canonical norm bounds every coefficient.

2.3 Homomorphic Encryption

CKKS [CKKS17] uses Ring-LWE encryption for approximate arithmetic on complex vectors. A vector $\mathbf{z} \in \mathbb{C}^{N/2}$ is encoded as $m = \lfloor \Delta \sigma^{-1}(\mathbf{z}) \rfloor$ for a scaling factor $\Delta > 0$, then encrypted with the primitive above. Decoding a phase μ gives $\sigma(\mu)/\Delta$, so encoding and encryption errors contribute to the approximation error. The scaling factor controls their size relative to the message, while the modulus must be large enough to avoid wrap-around.

GBFV [GV25] instead supports exact arithmetic in a plaintext ring $\mathcal{R}/t\mathcal{R}$, where $t = t(X)$ is a nonzero element of $\mathcal{R}$. A constant t recovers the plaintext ring of BFV [Bra12, FV12]. Suitable choices of t allow fewer plaintext slots with larger per-slot moduli, while keeping the coefficients of t small to limit noise growth. Thus, noise growth is governed by the coefficient size of the polynomial plaintext modulus t , rather than by the modulus of an individual slot.

In CKKS, addition with equal scaling factors is componentwise, while multiplication first forms the tensor $(c_0c'_0, c_0c'_1 + c_1c'_0, c_1c'_1)$, whose phase with respect to $(1, s, s^2)$ is the product of the input phases modulo q . The product also multiplies the scaling factors, so a rescaling by an integer factor $r \mid q$ rounds both ciphertext components after division by r , reducing the modulus to q/r and dividing the scaling factor by r [CKKS17]. Relinearization replaces the component under s^2 by two components under s , using keyswitching. Similarly, applying an automorphism τ_u to both components changes the secret to $\tau_u(s)$, and keyswitching restores s [GHS12a].

Keyswitching from s' to s decomposes $c_1 = \sum_i c_{1,i} B_i \bmod q$ into small digit polynomials $c_{1,i}$ with integer weights B_i . It uses public keys $\mathbf{k}_i = \text{Enc}_{pq}(s, pB_i s')$, where $p \geq 2$ is an auxiliary integer [BV11, GHS12b]. It returns

$$(c_0, 0) + \left\lfloor \frac{1}{p} \sum_i c_{1,i} \mathbf{k}_i \right\rfloor \bmod q.$$

The encrypted terms replace $c_1 s'$ under the new secret, while division by p reduces their accumulated error and introduces a coefficientwise rounding error. For relinearization, the same procedure is applied to the tensor's last component with $s' = s^2$.

3 The Proposed Scheme

We define the scheme for a public polynomial $g \in \mathcal{R}$ and a set of data slots, with the remaining slots serving as guards. The role of g is to control the error introduced in each slot without changing the integer ciphertext representation. We first define shaped rounding and the layouts, then encoding, encryption, and the homomorphic operations.

3.1 Shaped Rounding

For a ciphertext component $a \in \mathcal{R}$ and an integer divisor $r \geq 2$, ordinary coefficientwise rounding gives $\lfloor a/r \rfloor = (a - [a]_r)/r$. We replace the centered remainder $[a]_r$ by gh , where $g \in \mathcal{R}$ is a fixed public polynomial chosen small on the data slots, and h has bounded coefficients. The condition $gh \equiv a \pmod{r}$ ensures that the result still has integer coefficients, while the factor g controls the error in each slot.

Definition 1 (Shaped rounding). *Let $r \geq 2$ be an integer and $g \in \mathcal{R}$ be invertible in $\mathcal{R}_r$. For $a \in \mathcal{R}$, define*

$$h = \lfloor (g^{-1} \bmod r) a \rfloor_r, \quad \text{Round}_{g,r}(a) = \frac{a - gh}{r}.$$

Lemma 2. *The polynomial $\text{Round}_{g,r}(a)$ belongs to $\mathcal{R}$, and its error relative to a/r satisfies*

$$\text{Round}_{g,r}(a) - \frac{a}{r} = -g \frac{h}{r}, \quad \left\| \frac{h}{r} \right\|_{\infty} \leq \frac{1}{2}.$$

In particular, for every slot t ,

$$\sigma_t \left(\text{Round}_{g,r}(a) - \frac{a}{r} \right) = -\sigma_t(g) \sigma_t(h/r).$$

For $g = 1$, shaped rounding is ordinary coefficientwise rounding.

Proof. Since $gh \equiv a \pmod{r}$, every coefficient of $a - gh$ is divisible by r . The error identity follows from the definition, and the coefficient bound follows because h is centered modulo r . Applying the multiplicative map σ_t gives the slot identity. For $g = 1$, we have $h = [a]_r$, which gives the usual rounding with the tie convention fixed above. $\square$

By Lemma 1, the required invertibility is equivalent to $\gcd(\text{Norm}(g), r) = 1$. A unit g satisfies this condition for every integer factor r , so the same shaping polynomial can be used throughout the computation.

3.2 Layouts

A layout specifies a public shaping polynomial $g \in \mathcal{R}$, a set of data slots $S \subseteq \mathbb{Z}_{N/2}$, and a surjective map $\iota: S \rightarrow \{0, \dots, k-1\}$ identifying the value carried by each data slot. Several slots may carry copies of the same value, and the slots outside S are guards. We require g to be invertible modulo every ciphertext and evaluation-key modulus, and modulo the auxiliary encoding factor introduced below. This also ensures invertibility modulo every factor used for rescaling or keyswitching.

We call τ_u *admissible* for the layout if its slot permutation maps S onto itself, ensuring that no guard value is moved into a data slot.

The replicated layout places one data slot in every block of R slots and repeats the batch throughout the ring. A unit in a smaller ring gives the shaping polynomial the same value on every data slot.

Definition 2 (Replicated layout). *Let k and $R \geq 2$ be powers of two with $kR \leq N/2$, and put $d = N/(2kR)$ and $Y = X^{N/(2R)} = X^{kd}$. The data set and message indexing are*

$$S = \{t \in \mathbb{Z}_{N/2} : t \equiv 0 \pmod{R}\}, \quad \iota(t) = (t/R) \bmod k.$$

The shaping polynomial is $g = U(Y)$, where U is a unit of $\mathbb{Z}[Y]/(Y^{2R} + 1)$.

Each of the k message values has d copies, and each copy occupies one data slot followed by $R-1$ guards, as in Figure 1. Since U is a unit, g is a unit of $\mathcal{R}$ and meets the invertibility requirements for every modulus. We choose U with a small magnitude on the data class and controlled magnitudes on the guard classes.

Proposition 1 (Replicated layout properties). *For the layout of Definition 2, let $\omega = e^{i\pi/(2R)}$.*

1. *The value $\sigma_t(g) = U(\omega^{5^t})$ depends only on $t \bmod R$ and equals $U(\omega)$ on every data slot.*
2. *The admissible automorphisms are exactly $\tau_{\pm 5^j}$ with $R \mid j$. The rotation τ_{5^j} fixes g and sends the value at data index $\iota(t) + j/R$ to index $\iota(t)$, with indices taken modulo k . Conjugation preserves the data set and all magnitudes $|\sigma_t(g)|$, though it need not fix g .*
3. *For $\mathbf{z} \in \mathbb{C}^k$, the slot vector with value $z_{\iota(t)}$ on S and zero on the guards has period kR . Its inverse embedding lies in $\mathbb{R}[X^d] \subseteq \mathcal{R}_{\mathbb{R}}$, the real coefficient subring generated by X^d .*
4. *The trace onto K_d is $\text{Tr}_{K/K_d} = \sum_{i=0}^{d-1} \tau_{5^{kRi}}$. Every summand is admissible and fixes g , so the normalized trace fixes g and, on data slots, averages the d copies of the same message coordinate without mixing in guards.*

Proof. Evaluating Y at slot t gives $\sigma_t(Y) = \omega^{5^t}$. Since 5 has order R modulo $4R$, this depends only on $t \bmod R$ and equals ω on S . For $R \mid j$, we have $5^j \equiv 1 \pmod{4R}$, so $\tau_{5^j}(Y) = Y$ and the rotation fixes g . A rotation by j preserves S exactly when $R \mid j$, and then $\iota(t + j) = \iota(t) + j/R \bmod k$. Conjugation fixes every slot index and sends $\sigma_t(g)$ to its complex conjugate.

The intended slot values repeat every $kR = N/(2d)$ slots, including the zero guards. The projection formula from the preliminaries therefore leaves their inverse embedding unchanged and shows that only powers divisible by d occur. Finally, the trace formula follows from the same subring description. Its rotations are by multiples of kR , hence by multiples of R , and for $t \in S$ we have $\iota(t + kRi) = \iota(t)$. Each summand fixes g , and dividing their sum by d gives the normalized trace. $\square$

3.3 Encoding and Encryption

For a message $\mathbf{z} \in \mathbb{C}^k$, let $\tilde{\mathbf{z}} \in \mathbb{C}^{N/2}$ have entries $\tilde{z}_t = z_{\iota(t)}$ on S and zero elsewhere, and put $m = \sigma^{-1}(\tilde{\mathbf{z}})$. Encoding must also use shaped rounding, since ordinary coefficientwise rounding of Δm would introduce an unshaped error on the data slots before encryption. For a scaling factor $\Delta > 0$ and an integer encoding factor $r \geq 2$, first quantize with scaling factor $r\Delta$, then apply shaped rounding to divide it by r :

$$A = \lfloor r\Delta m \rfloor, \quad \text{Ecd}_{\Delta,r}^g(\mathbf{z}) = \text{Round}_{g,r}(A) \in \mathcal{R}.$$

The factor r is independent of the ciphertext modulus and controls the residual error from the initial quantization.

Indeed, writing $\nu = A - r\Delta m$ and $h = [(g^{-1} \bmod r)A]_r$, the encoded polynomial satisfies

$$\text{Ecd}_{\Delta,r}^g(\mathbf{z}) = \Delta m - g \frac{h}{r} + \frac{\nu}{r}, \quad \|h/r\|_{\infty} \leq \frac{1}{2}, \quad \|\nu/r\|_{\infty} \leq \frac{1}{2r}.$$

The first error term is shaped by g , while the unshaped term has canonical norm at most $N/(2r)$, since each slot is a sum of N coefficients multiplied by roots of unity. Increasing r makes its contribution to the decoded error, at most $N/(2r\Delta)$, as small as needed. The same encoding is used for plaintext operands in homomorphic operations.

Sample s from χ_{sk} as in the preliminaries, and fix one representative $t_j \in \iota^{-1}(j)$ for each message coordinate j . Private-key encryption and decryption with decoding are defined by the following operations.

- $\text{Enc}_{q,\Delta,r}^g(s, \mathbf{z})$ samples a uniformly in $\mathcal{R}_q$ and e from χ_e , and returns $\mathbf{c} = (-as + \text{Ecd}_{\Delta,r}^g(\mathbf{z}) + ge, a) \bmod q$.
 - $\text{Dec}_{q,\Delta}^g(s, \mathbf{c})$ computes $\mu = [c_0 + c_1 s]_q$ and returns $(\sigma_{t_j}(\mu)/\Delta)_{j=0}^{k-1}$.
- If $\|\text{Ecd}_{\Delta,r}^g(\mathbf{z}) + ge\|_\infty < q/2$, the phase of a fresh ciphertext is

$$\mu = \Delta m + g(e - h/r) + \nu/r.$$

Consequently, its error on a data slot $t \in S$ is

$$\frac{\sigma_t(\mu)}{\Delta} - z_{\iota(t)} = \frac{\sigma_t(g)}{\Delta} \sigma_t(e - h/r) + \frac{\sigma_t(\nu)}{r\Delta}.$$

On a guard slot the right-hand side gives the slot value itself, since the intended value is zero. Guard values are not returned by decoding, but they must remain bounded to prevent wrap-around during the computation.

3.4 Homomorphic Operations

The ciphertext arithmetic largely follows the CKKS blueprint, with usual rescaling and keyswitching replaced by their shaped versions. We extend $\text{Round}_{g,r}$ componentwise to ciphertexts.

Shaped Rescaling. Let $\mathbf{c} \in \mathcal{R}_q^2$ have scaling factor Δ , and let $r \geq 2$ divide q , with $q/r \geq 2$. Shaped rescaling returns

$$\mathbf{c}' = \text{Round}_{g,r}(\mathbf{c}) \bmod (q/r)$$

at modulus q/r and scaling factor Δ/r .

Let $\mu = [c_0 + c_1 s]_q$ and $h_i = [(g^{-1} \bmod r)c_i]_r$ for $i \in \{0, 1\}$. The phase of the ciphertext after shaped rescaling is

$$\mu' = [c'_0 + c'_1 s]_{q/r} = \left[\frac{\mu}{r} - g \frac{h_0 + h_1 s}{r} \right]_{q/r}.$$

Indeed, using the unreduced components $c'_i = (c_i - gh_i)/r$ and writing $c_0 + c_1 s = \mu + qI$ leaves only the multiple $(q/r)I$ to discard modulo q/r . If $\|\mu - g(h_0 + h_1 s)\|_\infty < q/2$, centered reduction has no effect, and the decoded slot values satisfy

$$\frac{\sigma_t(\mu')}{\Delta/r} = \frac{\sigma_t(\mu)}{\Delta} - \frac{\sigma_t(g)}{\Delta/r} \sigma_t\left(\frac{h_0 + h_1 s}{r}\right).$$

Shaped Keyswitching. To switch from s' to s , use the decomposition $c_1 = \sum_i c_{1,i} B_i \bmod q$ from the preliminaries and evaluation keys

$$\mathbf{k}_i = (-a_i s + p B_i s' + \varepsilon_i, a_i) \bmod pq,$$

where a_i is uniform in $\mathcal{R}_{pq}$ and $p \geq 2$ is the auxiliary modulus. Ordinary keys use $\varepsilon_i = e_i$ with e_i sampled from χ_e , while shaped keys use $\varepsilon_i = g e_i$. Set $\mathbf{D} = [\sum_i c_{1,i} \mathbf{k}_i]_{pq}$ and return

$$\mathbf{c}' = (c_0, 0) + \text{Round}_{g,p}(\mathbf{D}) \bmod q.$$

The ciphertext modulus and scaling factor remain q and Δ .

Let $\mu = [c_0 + c_1 s']_q$, and write $h_j = [(g^{-1} \bmod p) D_j]_p$ for $j \in \{0, 1\}$. The phase of the ciphertext after shaped keyswitching is

$$\mu' = [c'_0 + c'_1 s]_q = \left[\mu + \frac{1}{p} \sum_i c_{1,i} \varepsilon_i - g \frac{h_0 + h_1 s}{p} \right]_q.$$

Indeed, the phase of each key is $p B_i s' + \varepsilon_i$ modulo pq , and shaped division by p subtracts $g(h_0 + h_1 s)/p$. The decomposition identity then replaces $\sum_i c_{1,i} B_i s'$ by $c_1 s'$ modulo q . Centered reduction has no effect when the polynomial inside the brackets has coefficient norm less than $q/2$. The total keyswitching error consists of the evaluation-key noise contribution $p^{-1} \sum_i c_{1,i} \varepsilon_i$ and the rounding contribution $-g(h_0 + h_1 s)/p$. The rounding contribution is always shaped, and shaped evaluation keys also give a factor g in the noise contribution. With ordinary keys, the evaluation-key noise contribution must be bounded separately, as discussed in Section 6.4.

Addition and Multiplication. Ciphertexts at the same modulus and scaling factor are added componentwise, giving the sum of their phases modulo q without introducing a new error. To multiply ciphertexts at modulus q and scaling factors Δ and Δ' , form their tensor product as in the preliminaries, then relinearize by shaped keyswitching from s^2 to s . A shaped rescaling by $r \mid q$ gives a ciphertext at modulus q/r and scaling factor $\Delta \Delta' / r$. The output phase is the product of the input phases divided by r , plus the keyswitching error divided by r and the shaped rescaling error, followed by centered reduction modulo q/r .

A plaintext operand $a = \text{Ecd}_{\Delta_a, r_a}^g(\mathbf{w})$ multiplies both ciphertext components, without keyswitching. Following this product by shaped rescaling gives

$$\mathbf{c}' = \text{Round}_{g,r}(a\mathbf{c}) \bmod (q/r)$$

with scaling factor $\Delta \Delta_a / r$. The plaintext's encoding error contributes to the result, as does the shaped rescaling error.

Automorphisms. An automorphism τ_u is applied to both ciphertext components, followed by shaped keyswitching from $\tau_u(s)$ to s , leaving the modulus and scaling factor unchanged. Before keyswitching, the phase is $\tau_u(\mu)$ modulo q , so in

the absence of wrap-around the slot values are rotated and possibly conjugated as described in the preliminaries. Leveled computations use the admissible automorphisms defined in Section 3.2.

If $\tau_u(g) = g$, a shaped error remains shaped because $\tau_u(ge) = g\tau_u(e)$. More generally, preserving the magnitudes of the shaping profile only requires

$$|\sigma_t(\tau_u(g))| = |\sigma_t(g)| \quad \text{for every } t.$$

Conjugation satisfies this condition since $\sigma_t(\tau_{-1}(g)) = \overline{\sigma_t(g)}$, even if it does not fix g . Provided neither step wraps around, adding a ciphertext to its homomorphic conjugate and interpreting the sum with scaling factor 2Δ gives its real part, up to half the decoded error introduced by the conjugation's keyswitching.

Modulus Extension and Reduction. For an integer $\alpha \geq 1$, modulus extension sends c to αc at modulus αq and scaling factor $\alpha\Delta$. Its phase is exactly $\alpha\mu$, so the decoded slot values are unchanged and no error is introduced. Subsequent keyswitching requires evaluation keys at the enlarged modulus. Modulus reduction to a divisor $q' \geq 2$ of q instead reduces both components modulo q' without changing the scaling factor. Its phase is $[\mu]_{q'}$, which equals μ when $\|\mu\|_\infty < q'/2$.

3.5 Security Analysis

For fixed parameters, shaping does not weaken the underlying Ring-LWE assumption [LPR10]. Write $g\chi_e$ for the distribution of ge when e is sampled from χ_e .

Theorem 1. *Let $g \in \mathcal{R}$ be invertible modulo q . Decisional Ring-LWE with secret distribution χ_{sk} and error distribution $g\chi_e$ is equivalent to decisional Ring-LWE with the same secret distribution and error distribution χ_e , at the same ring dimension and modulus.*

Proof. The public map $(a, b) \mapsto (ga, gb)$ sends a Ring-LWE sample $(a, as + e)$ to $(ga, (ga)s + ge)$. Since multiplication by g is a bijection of $\mathcal{R}_q$, the first component remains uniform, and a uniform pair is mapped to a uniform pair. A distinguisher for the shaped samples therefore gives a distinguisher for the original samples with the same advantage. The inverse map $(a, b) \mapsto (g^{-1}a, g^{-1}b)$ gives the converse reduction. $\square$

Private-key encryption is therefore IND-CPA secure under this assumption. The reduction requires g to be invertible at every ciphertext and key modulus, including pq , as holds for units. Shaped rounding is public and introduces no additional secrecy assumption, while evaluation keys require the usual circular-security assumption as in CKKS. Shaped evaluation keys need no further assumption: multiplying both components of an ordinary key with message $g^{-1}pB_i s'$ by g gives a shaped key with message $pB_i s'$, since ga_i is uniform when a_i is. Their security therefore rests on the circular security of ordinary keys whose messages are multiplied by the public unit g^{-1} . By Theorem 1, concrete security can therefore be estimated as for CKKS with the same ring degree, modulus, secret distribution and error distribution χ_e .

Releasing Decrypted Results. IND-CPA security does not cover the release of approximate decryptions, whose errors depend on the secret key and can enable key recovery [LM21]. Our decoder releases only the data values, and guard slots must never be included in a released result. Protecting the data values requires the usual countermeasures, such as noise flooding calibrated to a bound on the released error and the statistical security target [LMSS22]. We discuss IND-CPA^D security and the effect of shaping on these countermeasures further in Appendix B.

4 Scheme Analysis

Correctness requires accurate data slots and bounded guards. Although guards are not decoded, their growth can cause coefficient wrap-around and invalidate the data-slot computation. Shaping makes the error injected on the data small, but multiplies the error injected on the guards by up to the largest guard magnitude of g , and the guards then take part in every subsequent operation. We first derive a sufficient scaling factor from guard stability and determine the data precision it provides (Section 4.1). We then extend this argument to general circuits (Section 4.2), derive practical parameters (Section 4.3), and bound what any shaping polynomial can achieve (Section 4.4).

4.1 Data Error, Guard Error and the Scaling Factor Law

Write $\mathbf{x} = \sigma(\mu)/\Delta$ for the vector of the slot values of a ciphertext, including its accumulated error. By Section 3.4, one shaped division by r adds $g\rho_r$ to the phase, with $\rho_r = -(h_0 + h_1s)/r$ and $\|h_i/r\|_\infty \leq 1/2$. Keyswitching adds $e_{\text{key}}^{\text{ks}} + g\rho_p$, where $e_{\text{key}}^{\text{ks}} = p^{-1} \sum_i c_{1,i} \varepsilon_i$ is the evaluation-key noise contribution and $g\rho_p$ is the error from its shaped division by p .

Table 1. Slotwise action of homomorphic operations. Both products include shaped rescaling, and the ciphertext product also includes relinearization.

Operation	Output in slot t
Addition with equal scaling factors	$x_t + x'_t$
Ciphertext product	$x_t x'_t + \eta_t$
Plaintext product	$x_t \tilde{w}_t + \eta_t$
τ_{5j} and shaped keyswitching	$x_{t+j} + \eta_t$
τ_{-5j} and shaped keyswitching	$\overline{x_{t+j}} + \eta_t$
Shaped rescaling	$x_t + \eta_t$
Realification	$\text{Re}(x_t) + \eta_t$
Modulus extension or reduction	x_t

Lemma 3 (Slotwise action of an operation). *Consider an operation in Table 1, with input slot values $\mathbf{x} = \sigma(\mu)/\Delta$ and, when applicable, $\mathbf{x}' = \sigma(\mu')/\Delta'$, and output scaling factor Δ_{out} . Assume that the unreduced phase expression of every ciphertext, including intermediate ciphertexts, has coefficient norm less than half the corresponding modulus. Then its output slot values are those in the table, where the injected error is*

$$\eta_t = \frac{\sigma_t(g)\sigma_t(\rho) + \sigma_t(e_{\text{key}})}{\Delta_{\text{out}}}.$$

Here ρ collects the shaped-rounding remainders, including those from keyswitching, and e_{key} collects only the evaluation-key noise contributions, each divided by any rescaling factors applied after it. For a plaintext operand a , encoded with scaling factor Δ_a , we write $\tilde{w}_t = \sigma_t(a)/\Delta_a$.

Proof. Each row follows from the phase identities of Section 3.4, divided by the output scaling factor. For a ciphertext product, the output phase is $(\mu\mu' + e_{\text{key}}^{\text{ks}} + g\rho_p)/r + g\rho_r$ with scaling factor $\Delta_{\text{out}} = \Delta\Delta'/r$, which gives $\rho = \rho_p/r + \rho_r$ and $e_{\text{key}} = e_{\text{key}}^{\text{ks}}/r$. A plaintext product has $\rho = \rho_r$ and $e_{\text{key}} = 0$. Automorphisms permute and possibly conjugate the slots as in Section 2, and realification reads the sum of a ciphertext and its conjugate with scaling factor 2Δ . The no-wrap assumption ensures that centered reduction leaves each phase unchanged. $\square$

A normalized trace onto K_d , with $n = N/(2d)$, adds the current ciphertext to its rotation by $2^i n$ slots for $0 \leq i < \log_2 d$ and reads the result with scaling factor $d\Delta$. When these rotations are admissible, it returns the orbit average $d^{-1} \sum_{i=0}^{d-1} x_{t+ni}$ plus the accumulated keyswitching error divided by $d\Delta$. The error of step i is copied into $d/2^{i+1}$ summands, so the decoded error is at most $1 - 1/d$ times a common bound for one keyswitching at the input scaling factor. For traces and realifications, the no-wrap condition must also hold for every intermediate ciphertext.

Error Bounds for Each Operation. For an operation f , let E_f and $E_{\text{key},f}$ bound $|\sigma_t(\rho)|$ and $|\sigma_t(e_{\text{key}})|$ in every slot, and write

$$m_S = \max_{t \in S} |\sigma_t(g)|, \quad M = \max_{t \notin S} |\sigma_t(g)|,$$

with $M = 0$ if there are no guard slots. The operation then injects at most

$$\delta_f = \frac{m_S E_f + E_{\text{key},f}}{\Delta_{\text{out}}} \quad \text{on data slots,} \quad c_f = \frac{M E_f + E_{\text{key},f}}{\Delta_{\text{out}}} \quad \text{on guard slots.}$$

A plaintext operand also carries its encoding error, multiplied by $|x_t|$ when bounding the error of the product against the intended plaintext. With H_a and H'_a bounding $\|h/r_a\|_{\text{can}}$ and $\|\nu/r_a\|_{\text{can}}$ from Section 3.3, its slot value differs from the intended one by at most $(|\sigma_t(g)|H_a + H'_a)/\Delta_a$, with the deterministic bounds $H_a \leq N/2$ and $H'_a \leq N/(2r_a)$.

Squaring and Stability. Consider a chain of squarings with a fixed scaling factor Δ , where E bounds the aggregate shaped-rounding remainder from relinearization and rescaling, and E_{key} bounds the evaluation-key noise contribution, in the output phase. Each squaring injects at most $c = (ME + E_{\text{key}})/\Delta$ on the guards, so a guard of modulus at most b becomes one of modulus at most $b^2 + c$.

Lemma 4 (Squaring chains). *Assuming no coefficient wrap-around, a disk $B(b) = \{z \in \mathbb{C} : |z| \leq b\}$ is invariant under the squaring chain whenever $b^2 + c \leq b$. For $c \leq 1/4$, this holds for every $b \in [b^*, b^+]$, where*

$$b^* = \frac{1 - \sqrt{1 - 4c}}{2}, \quad b^+ = \frac{1 + \sqrt{1 - 4c}}{2}.$$

If the initial guards lie in $B(b_0)$ with $0 \leq b_0 < b^+$, they remain in $B(\max(b_0, b^*))$ and admit radius bounds converging to $b^* \leq 1/2$. For $c > 1/4$, the worst-case recurrence $b_{j+1} = b_j^2 + c$ diverges.

Proof. Squaring a value in $B(b)$ and adding an error in $B(c)$ gives a value in $B(b^2 + c)$. The map $b \mapsto b^2 + c$ is increasing on $[0, \infty)$, lies below the identity between its fixed points b^* and b^+ when $c \leq 1/4$, and above it below b^* , so the recurrence started below b^+ converges monotonically to b^* . For $c > 1/4$, $b_{j+1} - b_j = (b_j - 1/2)^2 + c - 1/4 \geq c - 1/4 > 0$. $\square$

The lemma gives the sufficient condition on the scaling factor

$$\Delta \geq 4(ME + E_{\text{key}}),$$

together with two explicit requirements. The initial guards must lie in a disk of radius below b^+ . For fresh encryptions this is checked from the encoding and encryption errors of Section 3.3, and Section 5.3 gives the corresponding condition after bootstrapping. At every step, including intermediate ciphertexts, the scaling factor times the largest data and guard values must remain below half the modulus, so that no phase wraps around. Above $c = 1/4$, the disk argument gives no guarantee, but it does not imply that actual rounding errors make the guards grow.

The Scaling Factor Law. In CKKS, keeping b_{prec} bits of precision after an operation with error bound E requires roughly $\log_2 \Delta \geq b_{\text{prec}} + \log_2 E$. Here the stability condition instead reads

$$\log_2 \Delta \geq 2 + \log_2 (ME + E_{\text{key}}) \approx 2 + \log_2 M + \log_2 E$$

when the evaluation-key noise contribution is negligible. With this scaling factor, a level adds at most $\delta = (m_S E + E_{\text{key}})/\Delta$ on the data. Stability gives a sufficient scaling factor, while m_S determines the resulting data precision. The target precision still constrains the shaping polynomial, evaluation-key noise contribution and shaping depth, and the circuit analysis must account for subsequent error amplification. A product followed by rescaling back to Δ consumes

approximately $\log_2 \Delta$ modulus bits. The nominal saving over CKKS is about $b_{\text{prec}} - 2 - \log_2 M$ bits per level, before margins for the failure probability and error accumulation.

The unit of our implementation (Section 6.1) has $m_S \approx 2^{-39.7}$ and $M \approx 2^{4.7}$ at $R = 16$, so the saving is about 13 bits for $b_{\text{prec}} = 20$. With the rounding-error size $E \approx 2^{13.3}$ measured at $N = 2^{15}$, the fully shaped data-error estimate with scaling factor 2^{20} is $2^{-46.4}$ when the evaluation-key noise contribution is negligible, against $2^{-6.7}$ for CKKS with the same scaling factor. This is the error of a leveled operation, not the precision of a bootstrapped ciphertext, which also includes bootstrapping errors. Section 6.4 gives the representation-dependent choice of E_{key} . When an implementation shapes only part of each rounding, the data error gains the unshaped term of Lemma 10 in Section 6.3.

4.2 Correctness of General Circuits

The squaring argument follows one disk through repeated applications of one operation. We represent each homomorphic operation, including normalized traces and realifications, as a gate in the circuit together with its associated keyswitching and rescaling steps. General circuits mix products, plaintext operands, rotations and traces, so we attach to each wire a set of possible guard values and measure the data error against the intended computation. The sets must contain complex values even when the data is real, since rounding errors have complex slot values.

Definition 3 (Data error and guard domain). *A ciphertext with slot values $\mathbf{x} = \sigma(\mu)/\Delta$ encrypts $\mathbf{z} \in \mathbb{C}^k$ with data error at most ε and guard domain $\Gamma \subseteq \mathbb{C}$ if*

$$|x_t - z_{\iota(t)}| \leq \varepsilon \quad \text{for } t \in S, \quad x_t \in \Gamma \quad \text{for } t \notin S.$$

Recall that $B(r) = \{z \in \mathbb{C} : |z| \leq r\}$ is the closed disk centered at zero with radius $r \geq 0$. For a gate f , write F_f for its slot operation before the injected error, as in Table 1, applied to sets of values. Addition and multiplication use the set operations $\Gamma_1 + \Gamma_2 = \{u + v : u \in \Gamma_1, v \in \Gamma_2\}$ and $\Gamma_1 \Gamma_2 = \{uv : u \in \Gamma_1, v \in \Gamma_2\}$, respectively. A product by a plaintext a gives $\Gamma \Pi_a$, for a set Π_a containing the guard values $\tilde{w}_t$ of a . Admissible rotations preserve Γ , conjugation and realification map it to $\overline{\Gamma}$ and $\text{Re}(\Gamma)$, a normalized trace maps it into its convex hull $\text{conv}(\Gamma)$, and modulus extension and reduction leave it unchanged with $c_f = 0$. Plaintexts encoded with zero guards have Π_a bounded by their encoding error, which can make $\Gamma \Pi_a$ much smaller than Γ .

Definition 4 (Perturbed evaluation). *Fix the input data-slot values of a circuit. A perturbed evaluation applies the slot operations F_f gate by gate, with plaintext operands entering through their encoded values $\tilde{w}_t$, and adds at each output data slot a perturbation of magnitude at most δ_f .*

The usual CKKS error analysis bounds a perturbed evaluation, with δ_f replacing the error added by each gate. Admissibility ensures that its data-slot

operations never read guard values, so it remains to keep every phase within its modulus.

Theorem 2 (Guard-domain stability). *Consider a circuit whose gates implement the operations of Lemma 3 and normalized traces, with layout (g, S, ι) and only admissible automorphisms, whose injected errors satisfy the bounds δ_f and c_f . Assign each wire w a scaling factor Δ_w , a modulus q_w , a guard domain $\Gamma_w \subseteq \mathbb{C}$ and a data bound $D_w \geq 0$, and assume the following conditions.*

1. *For every gate f with input wires $w_1, \dots, w_\ell$ and output wire w ,*

$$F_f(\Gamma_{w_1}, \dots, \Gamma_{w_\ell}) + B(c_f) \subseteq \Gamma_w.$$

2. *Every perturbed evaluation from the input data-slot values has data values of modulus at most D_w on each wire w .*
3. *Every wire satisfies $\Delta_w \max(D_w, \sup_{z \in \Gamma_w} |z|) < q_w/2$, and the same inequality holds for every intermediate ciphertext inside a gate, with its own scaling factor, modulus and propagated bounds.*
4. *Each input ciphertext has its guard values in the domain of its wire.*

Then every wire has its guard values in Γ_w , and its data values are those of a perturbed evaluation from the input data-slot values.

Proof. We proceed in circuit order, the claim holding on the inputs by assumption. Consider a gate f whose inputs satisfy the claim, and let μ_{out} be its output phase expression before centered reduction, as in the proof of Lemma 3. Since admissible automorphisms keep data and guards separate, the data slots of $\mu_{\text{out}}/\Delta_w$ extend the input perturbed evaluation by one gate, so they are bounded by D_w . Its guard slots lie in $F_f(\Gamma_{w_1}, \dots, \Gamma_{w_\ell}) + B(c_f) \subseteq \Gamma_w$, using the encoded guard values for a plaintext product and the convex hull for a trace. Hence $\|\mu_{\text{out}}\|_\infty \leq \|\mu_{\text{out}}\|_{\text{can}} \leq \Delta_w \max(D_w, \sup_{z \in \Gamma_w} |z|) < q_w/2$, and the same holds at each intermediate step, so Lemma 3 identifies these values with the actual output slots. $\square$

Any precision bound for the perturbed evaluation therefore bounds the data error of the ciphertexts, and together with Γ_w it gives the invariant of Definition 3 on each wire. Lemma 4 is the special case of disks and squarings. Products by fresh operands are the other simple case, and they contract the guards.

Corollary 1 (Products by fresh operands). *Suppose each multiplication uses a fresh operand whose guards lie in $B(b_f)$, with $0 \leq b_f < 1$, and injects at most $c \geq 0$ on the guards. Then $B(b)$ is invariant for every $b \geq c/(1 - b_f)$, and guards initially in $B(b_0)$ remain in $B(\max(b_0, c/(1 - b_f)))$.*

Proof. A product maps $B(b)B(b_f) + B(c)$ into $B(bb_f + c)$, which lies in $B(b)$ exactly when $bb_f + c \leq b$. $\square$

Polynomial Computations. For a polynomial $P(Z) = \sum_i a_i Z^i$ with real coefficients, write $P_{\text{abs}}(b) = \sum_i |a_i| b^i$, and let δ_P bound the error of its evaluation circuit on the guard domain. A disk $B(b)$ is invariant when $P_{\text{abs}}(b) + \delta_P \leq b$, but no positive b satisfies this if $a_0 = 0$ and $|a_1| > 1$. Activations with $P(0) = 0$ and slope greater than one are of this kind. The following operations can help control their guards. A product by a fresh operand contracts the guards as in Corollary 1. Realification places the guards near a real interval, with a complex error from keyswitching, but repeated polynomial evaluation can still amplify their values. Appendix C gives the corresponding interval condition, and Section 7.3 reports measurements of the three tools on a degree-seven activation.

Guard Reset. A guard reset multiplies the ciphertext by a shaped encoding a of one on the data slots and zero on the guards, then applies shaped rescaling. It preserves the data up to encoding and rounding errors and sends a guard bound b to at most $(MH_a + H'_a)b/\Delta_a + c_{\text{mask}}$, where c_{mask} bounds the rescaling error and the encoding bounds are as in Section 4.1. The costs is one plaintext product and one rescaling. When the circuit already multiplies by an encoded plaintext, setting its intended guard values to zero incorporates the reset into that product, with no additional level if its encoding precision and rescaling suffice.

4.3 Practical Scaling Factor Selection

The deterministic bound $|\sigma_t(\rho_r)| \leq \frac{N}{2}(1 + |\sigma_t(s)|)$, with $|\sigma_t(s)| \leq h_{\text{wt}}$ for a sparse ternary secret of Hamming weight h_{wt} , gives $E \approx 2^{24}$ at $N = 2^{15}$ and $h_{\text{wt}} = 1024$. The resulting threshold $4ME \approx 2^{30.7}$ lies almost 12 bits above the measured stability edge of our unit. The worst-case bound ignores cancellation among the N terms in each slot evaluation.

Heuristic Noise Model. Assume that the coefficients of h_0/r and h_1/r behave as independent uniform variables in $[-1/2, 1/2)$, independently between levels, and model their slot values as centered complex Gaussians. For a fixed secret and negligible evaluation-key noise contribution, the error injected on slot t by a squaring with scaling factor Δ then has standard deviation

$$\gamma_t(s) = \frac{|\sigma_t(g)|}{\Delta} \sqrt{\frac{N(1 + |\sigma_t(s)|^2)}{12}}.$$

Substituting standard deviations into the guard-stability condition suggests the practical threshold

$$\Delta \gtrsim 4 \max_{t \notin S} |\sigma_t(g)| \sqrt{\frac{N(1 + |\sigma_t(s)|^2)}{12}},$$

which is not a correctness guarantee, since the Gaussian model has unbounded tails. The most exposed slots are those where both $|\sigma_t(g)|$ and $|\sigma_t(s)|$ are large, and they remain the same throughout a computation because the secret is fixed. For our unit at $N = 2^{15}$ and one ternary secret of weight 1024, this threshold is

near $2^{18.8}$. Our implementation diverges with scaling factor 2^{18} and remains stable over sixteen squarings at 2^{19} and 2^{20} . Section 7.3 reports the same agreement for other units and ring degrees.

Failure Probability. The model locates the transition, but rare failures need a separate estimate. We estimate escape through sequences of injected errors that drive a guard beyond radius 1.2, after which squaring amplifies its magnitude. Under the Gaussian model, the minimum-cost path from an initial radius b_0 gives the exponential estimate $\exp(-J(b_0)/\gamma_t(s)^2)$, with the cost $J(b_0)$ computed numerically in Appendix C.2. Summing over guard slots and starting levels, and averaging over the secret distribution, which sets $\gamma_t(s)$, gives the estimates of Table 2.

Table 2. Heuristic guard-escape probabilities over 21 squarings, averaged over the secret distribution under the Gaussian model. The columns give the assumed initial guard radius.

$\log_2 \Delta$	$b_0 = 0$	$b_0 = 0.2$	$b_0 = 0.37$
19	$2^{-11.5}$	$2^{-10.7}$	$2^{-8.3}$
20	$2^{-39.4}$	$2^{-37.7}$	$2^{-33.0}$
21	$2^{-95.6}$	$2^{-92.3}$	$2^{-82.8}$
22	$2^{-208.6}$	$2^{-202.0}$	$2^{-182.8}$

The 20-bit levels of our measurements give estimates between 2^{-33} and 2^{-39} , while 22-bit levels give estimates below 2^{-180} for the same horizon. Guard escape can eventually invalidate the no-wrap condition and corrupt the released data, even though the guards themselves are not decoded. Under this heuristic, we therefore recommend 22-bit levels for negligible probability of failure. They cost $2L$ additional modulus bits for L levels, reducing the supported depth at $N = 2^{15}$ from 21 to about 19 in our parameter sets, and without a performance cost.

A 200 bootstrapping experiment for tail measurement, each followed by 16 squarings, reach a largest guard magnitude of $2^{-1.44} \approx 0.37$, which motivates the last column. This is close to the stationary radius $b^* \approx 0.35$ of Lemma 4 for the measured guard error $c = 2^{-2.13}$ injected by one shaped rounding, but it is not a certification of the rare-event probabilities. These estimates concern the leveled computation, and Section 5.4 treats failures within bootstrapping.

4.4 Limits of Shaping

Reducing m_S improves data precision, while increasing M raises the scaling factor needed to keep the guards stable. These two quantities cannot be chosen independently because a nonzero integer polynomial has a nonzero integer algebraic norm.

Theorem 3 (Norm bound). *Let $g \in \mathcal{R} \setminus \{0\}$ and $\emptyset \neq S \subsetneq \mathbb{Z}_{N/2}$. With $m_S = \max_{t \in S} |\sigma_t(g)|$ and $M = \max_{t \notin S} |\sigma_t(g)|$,*

$$\log_2 \frac{1}{m_S} \leq \left(\frac{N}{2|S|} - 1 \right) \log_2 M.$$

Proof. The embeddings come in conjugate pairs, so

$$1 \leq |\text{Norm}(g)| = \prod_{t \in \mathbb{Z}_{N/2}} |\sigma_t(g)|^2 \leq m_S^{2|S|} M^{N-2|S|}.$$

Taking base-two logarithms and rearranging gives the bound. $\square$

In the replicated layout of Definition 2, $|S| = kd = N/(2R)$, so the bound becomes $\log_2(1/m_S) \leq (R-1) \log_2 M$. Each bit of guard amplification can therefore buy at most $R-1$ bits of error reduction on the data slots, where $|S|$ counts all data slots, including replicated copies.

For a unit g , the product of the embedding magnitudes is exactly one, giving

$$\sum_{t \in S} \log_2 \frac{1}{|\sigma_t(g)|} = \sum_{t \notin S} \log_2 |\sigma_t(g)|.$$

For a non-unit, the positive contribution of $\log_2 |\text{Norm}(g)|$ leaves less room for reducing the data-slot error at the same guard bound. For $g = U(Y)$ in the replicated layout, Proposition 1 reduces the identity to

$$\log_2 \frac{1}{|U(\omega)|} = \sum_{j=1}^{R-1} \log_2 |U(\omega^{5^j})| \leq (R-1) \log_2 M,$$

with equality when all guard classes have magnitude M . This motivates units with a nearly constant guard profile, which Section 6.1 searches for. The unit of our implementation has $\log_2(1/m_S) \approx 39.7$, below the bound $15 \log_2 M \approx 70.5$, because its guard classes average about 2.65 bits rather than $\log_2 M \approx 4.7$.

5 Bootstrapping Our Scheme

Bootstrapping must recover the data at a larger modulus while returning guards small enough for the next leveled computation. We follow the CKKS bootstrapping blueprint [CHK⁺18] in SlotsToCoeffs-first order, with linear transformations between slots and coefficients and an approximate modular reduction. The transforms must exclude the incoming guard values from the recovered data, and their evaluation must preserve data precision even though intermediate rotations mix the slot classes. Returning from EvalMod to the leveled scaling factor can amplify guard errors beyond the stability bounds of Section 4, so the last step must control this amplification.

The central step, ModUp, reinterprets ciphertext components at a larger modulus without scaling them. If their phase at the bottom modulus q_b is μ , the

lifted phase has the form $\mu + q_b I$ for some integer polynomial I . `CoeffsToSlots` exposes its coefficients in the slots, so that `EvalMod` can approximately remove the multiples of q_b . We use sparse-secret encapsulation around `ModUp` [BTH22], temporarily switching to a sparse secret to control the size of I .

For the replicated layout, let $d = N/(2kR)$ as before and assume $R \mid k$, padding a smaller batch to R values when necessary. The bootstrapping algorithm proceeds as follows.

1. Apply the normalized bottom trace onto K_d , using shaped keyswitching and rounding. It averages the replicated copies of each data value without reading guards, as in Proposition 1.
2. Apply the fiber-aware `SlotsToCoeffs` transform, whose map only reads data slots. It is chosen so that, after the top trace below, the data appear as coefficients in the subring of degree $2k$.
3. Raise the modulus with `ModUp` and sparse-secret encapsulation.
4. Apply the normalized top trace onto $K_{N/(2k)}$, followed by the fiber-aware `CoeffsToSlots` transform. The trace retains the desired coefficient message and an integer multiple of q_b , while the transform writes the resulting values to the replicated data class with zero on the guards in the exact map. Separate the real and imaginary parts for modular reduction.
5. Apply `EvalMod` separately to the real and imaginary parts to remove the lifted multiples of q_b , keeping its constants zero on the guards.
6. Return to the leveled scaling factor with bounded guards and recombine the real and imaginary outputs.

We first construct the two transforms, then bound their evaluation errors using extra working precision. The final step is to control the guard errors introduced by `EvalMod` and the return to the leveled scaling factor.

5.1 Fiber-Aware Linear Transformations

`SlotsToCoeffs` must place the k data values in the coefficient message without importing the guards. Although its input has period kR , the pipeline only needs the desired message after the top trace onto $K_{N/(2k)}$. We use this freedom to distribute each transformed value among the positions that the trace averages, with every contribution reading a data slot. This reduces the construction to k diagonals.

Put $n = kR$ and assume $R \mid k$ as in the pipeline. On a slot vector of period n , the normalized top trace averages the R positions $c + ki$ for $0 \leq i < R$, for each $0 \leq c < k$. We call these positions the fiber of c . Write $\text{rot}_o(\mathbf{x})_t = x_{t+o}$, with indices modulo n , and let $\odot$ denote componentwise multiplication. A diagonal $\mathbf{D}_o$ specifies the plaintext weights applied to the rotation by o . Figure 2 illustrates one fiber for $k = 4$ and $R = 2$.

Lemma 5 (Counting lemma). *Let $\mathbf{x} \in \mathbb{C}^n$ have data entries $z_j = x_{Rj}$ for $0 \leq j < k$, with arbitrary entries on the guards, and set*

$$O = \{-t + Ru \bmod n : 0 \leq t < R, 0 \leq u < k/R\}.$$

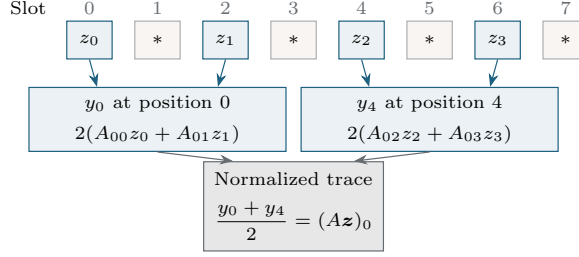


Fig. 2. Computing $(Az)_0$ for a transform matrix A with $k = 4$ and $R = 2$. The two fiber positions 0 and 4 collect disjoint parts of the desired sum, with weights multiplied by $R = 2$ to compensate for the normalized trace. The guards (*) are not read, and the other output fibers are omitted.

The set O has k elements. For every $A \in \mathbb{C}^{k \times k}$, there are diagonals $(D_o)_{o \in O}$ with $D_o[s] = 0$ whenever $s + o \not\equiv 0 \pmod{R}$, such that

$$\mathbf{y} = \sum_{o \in O} D_o \odot \text{rot}_o(\mathbf{x}) \quad \text{satisfies} \quad \frac{1}{R} \sum_{i=0}^{R-1} y_{c+ki} = (Az)_c \quad (0 \leq c < k).$$

Proof. The residue of an offset $-t + Ru$ modulo R determines t , and its residue modulo n then determines u in the specified range. Thus the $R(k/R) = k$ pairs give distinct offsets.

Fix a fiber c and write its positions as $s_i = c + ki$. Since $R \mid k$, the offset $-t + Ru$ reads a data slot at s_i exactly when $t = c \pmod{R}$. For this value of t , it reads data index

$$j(i, u) = \left\lfloor \frac{c}{R} \right\rfloor + \frac{k}{R}i + u \pmod{k}.$$

As (i, u) ranges over $0 \leq i < R$ and $0 \leq u < k/R$, the index $j(i, u)$ takes every value in $\mathbb{Z}_k$ exactly once. Set $D_o[s_i] = RA_{c, j(i, u)}$ for $o = -(c \pmod{R}) + Ru$, and set the remaining diagonal entries at s_i to zero. Then

$$\frac{1}{R} \sum_{i=0}^{R-1} y_{c+ki} = \frac{1}{R} \sum_{i=0}^{R-1} \sum_{u=0}^{k/R-1} RA_{c, j(i, u)} z_{j(i, u)} = \sum_{j=0}^{k-1} A_{c, j} z_j,$$

which proves the claim. $\square$

Taking A to be the sparse SlotsToCoeffs matrix gives the forward transform in the pipeline. Each position in a fiber collects k/R terms, and the trace combines the R partial sums to recover the full transform. The factor R in the diagonal weights compensates for the normalization of the trace. The diagonals vanish wherever a rotation would read a guard, so the exact map is independent of the guard values. The effect of quantization and evaluation errors is treated in Section 5.2.

Lemma 6 (Gather bound). *Let $R \geq 2$ and k be powers of two, with $n = kR$. Any sum of rotated diagonals of period n , zero wherever they read a guard, whose normalized trace returns the period- k replication of $\mathbf{z}$ for every input with $x_{Rj} = z_j$, needs at least k distinct offsets modulo n .*

Proof. Take $\mathbf{z}$ to be the unit vector at c and set the guards to zero. Since the output at c is one, some nonzero diagonal entry must read slot Rc from a position $c + ki$, giving $c + ki + o \equiv Rc \pmod{n}$, hence $o \equiv (R - 1)c \pmod{k}$. As $R - 1$ is odd, these residues cover $\mathbb{Z}_k$, requiring at least k offsets. $\square$

Thus Lemma 6 shows that the counting lemma is optimal: gathering already requires k diagonals in a single transform, even for the identity. Factoring the gather would require controlling errors in the intermediate layouts.

The Reverse Transform. CoeffsToSlots must expose the coefficient message for EvalMod while restoring the replicated layout. After the top trace its input has period k , so k diagonals suffice to write the data class with zero guard rows.

Lemma 7 (Period- k inputs). *For $\mathbf{x} \in \mathbb{C}^n$ of period k , every complex-linear map to $\mathbb{C}^n$ can be evaluated with at most k diagonals as*

$$\mathbf{y} = \sum_{o=0}^{k-1} \mathbf{D}_o \odot \text{rot}_o(\mathbf{x}).$$

Proof. For $y_s = \sum_{j=0}^{k-1} A_{s,j} x_j$, take $D_o[s] = A_{s, (s+o) \bmod k}$, since $\text{rot}_o(\mathbf{x})_s = x_{(s+o) \bmod k}$. $\square$

CoeffsToSlots produces $v_j = m_j + im_{j+k}$ from the real coefficient values m_j . Conjugation separates $(\mathbf{v} + \bar{\mathbf{v}})/2$ and $(\mathbf{v} - \bar{\mathbf{v}})/(2i)$ for EvalMod, whose outputs are then recombined. The transform and this separation are evaluated at the higher working precision described next. Our implementation uses baby-step giant-step evaluation [HS14].

5.2 Precision During the Linear Transforms

The forward map excludes the guard slots, but this alone does not protect against errors introduced while evaluating it. The rotations used by both transforms need not preserve the data class, so a shaped error introduced at an intermediate step can later reach a data slot through its large guard components. We therefore use ordinary internal roundings and suppress their errors by evaluating with a larger intermediate scaling factor.

SlotsToCoeffs ends in the coefficient domain, where shaping does not suppress the error that CoeffsToSlots will later expose, so its final rounding is ordinary as well. After ModUp, the top trace and CoeffsToSlots take a coefficient message back to the replicated layout, including separation of the real and imaginary parts for EvalMod. Only at this output do we use a shaped rounding.

A *wide stage* with an integer $w \geq 0$ extra bits starts with an exact modulus extension by 2^w , or retains these bits by shortening a preceding rescale. Its intermediate scaling factors and moduli carry this extra factor, and a final division by 2^w returns to the intended output scaling factor. For $w > 0$, this final division uses shaped rounding when the output is in the replicated slot layout, and ordinary rounding at the SlotsToCoeffs output. Extension and final division cancel, so the extra bits cause no net modulus consumption beyond the stage’s own products, but they increase the largest ciphertext and evaluation-key moduli that must meet the security limits.

Lemma 8 (Wide-stage error). *Let a wide stage evaluate a real-linear map A on slot vectors and finish with scaling factor Δ_{out} . Write $\|A\|_{\infty \rightarrow \infty} = \sup_{\|v\|_{\infty} \leq 1} \|Av\|_{\infty}$, and suppose the input error is at most ε_{in} on every slot read by A . Let E_{int} bound the total internal error in unnormalized slot values immediately before the final division, after propagating each contribution through the subsequent operations up to that point. If the final shaped phase remainder is $g\rho$ with $|\sigma_t(\rho)| \leq E_{\text{rnd}}$, and no coefficient wrap-around occurs, the output error in slot t is at most*

$$\|A\|_{\infty \rightarrow \infty} \varepsilon_{\text{in}} + \frac{E_{\text{int}}}{2^w \Delta_{\text{out}}} + \frac{|\sigma_t(g)| E_{\text{rnd}}}{\Delta_{\text{out}}}.$$

For $w = 0$, the final division is omitted and $E_{\text{rnd}} = 0$.

Proof. Let $\mathbf{x}$ be the intended input and $\mathbf{e}$ its error, setting the entries of $\mathbf{e}$ to zero on slots not read by A . Immediately before the final division, the phase has slot vector

$$2^w \Delta_{\text{out}} A(\mathbf{x} + \mathbf{e}) + \boldsymbol{\xi}, \quad \|\boldsymbol{\xi}\|_{\infty} \leq E_{\text{int}}.$$

Division by 2^w , followed by normalization by Δ_{out} , gives the error

$$A\mathbf{e} + \frac{\boldsymbol{\xi}}{2^w \Delta_{\text{out}}} + \frac{\sigma(g\rho)}{\Delta_{\text{out}}}.$$

The triangle inequality and the operator-norm bound give the claim. $\square$

The three terms separate propagated input error, errors introduced during evaluation, and the final rounding. Extra working bits suppress only the second term, which includes rounding, evaluation-key noise and plaintext-quantization errors, including the effect of quantized diagonals on guard values. They cannot remove guard values read by the exact map, which is why the forward transform excludes those inputs. For SlotsToCoeffs, whose final rounding is ordinary, the same argument replaces $|\sigma_t(g)| E_{\text{rnd}}$ by the corresponding unshaped phase-error bound. The resulting slot-error bound also bounds the coefficient error, since $\|\cdot\|_{\infty} \leq \|\cdot\|_{\text{can}}$.

5.3 EvalMod and Its Constants

CoeffsToSlots writes zero guards in exact arithmetic, but a constant term in EvalMod would populate them again. We keep the intended guards at zero by

encoding each real plaintext constant a as $a\pi_S$, where $\pi_S = \sigma^{-1}(\mathbf{1}_S)$ is the data-slot indicator, and use shaped products and constant encodings to preserve data precision. This incorporates the guard mask into the polynomial evaluation. Since the indicator has period R , $\pi_S \in \mathbb{R}[Y]$, with $Y = X^{N/(2R)}$.

Proposition 2 (Compact shaped constants). *For $x \in \mathbb{R}[Y]$, $\Delta_a > 0$ and an integer $r_a \geq 2$,*

$$a_x = \text{Round}_{g, r_a}(\lfloor r_a \Delta_a x \rfloor) \in \mathbb{Z}[Y],$$

with at most $2R$ nonzero coefficients.

Proof. Quantization preserves support on powers of Y , and $g, g^{-1} \in \mathbb{Z}[Y]$ because U is a unit. The remainder $h = \lfloor (g^{-1} \bmod r_a) \lfloor r_a \Delta_a x \rfloor \rfloor_{r_a^q}$ and the quotient defining a_x therefore stay in this subring, which has $2R$ coefficient positions. $\square$

A product by $a\pi_S$ clears the guards in exact arithmetic, while an addition changes only the data, with encoding errors bounded as in Section 4.1. Each constant product costs $O(RN)$ coefficient operations. The subsequent double-angle squarings still introduce guard errors.

Restoring Bounded Guards. Let $\beta = q_b/\Delta_b$, where Δ_b is the scaling factor of the coefficient message before ModUp, and let Δ_{em} be the EvalMod scaling factor. Returning from normalized EvalMod values to message units multiplies errors by β , so a shaped rounding of size E , amplified by G_{em} through the remaining operations, contributes approximately

$$\frac{|\sigma_t(g)|EG_{\text{em}}\beta}{\Delta_{\text{em}}}.$$

This can exceed one on the guards despite being negligible on the data.

To reduce the guard error before this amplification, we use the last double-angle squaring, whose masked additive constant leaves only the quadratic map $v \mapsto 2v^2$ on a guard, with the factor two applied exactly. We leave this squaring unrescaled and combine its division with the return to the leveled scaling factor Δ , so its rounding is introduced at the final scaling factor. A guard of magnitude at most ε_{em} therefore contributes at most $2\beta\varepsilon_{\text{em}}^2$ at the output, plus the constant-encoding errors and

$$c_{\text{final}} = \frac{ME_{\text{final}} + E_{\text{key, final}}}{\Delta}$$

from relinearization and the final shaped rescale. Here E_{final} bounds the aggregate shaped-rounding remainder of these two operations, and $E_{\text{key, final}}$ bounds only the evaluation-key noise contribution, both after propagation to the final phase. In particular, the rounding introduced by division by p during relinearization is included in E_{final} , with the subsequent final rescaling applied to it. When ε_{em} is proportional to $1/\Delta_{\text{em}}$, each extra working bit reduces the quadratic term by two bits, until the final-rescale error dominates. The total must meet the initial guard condition of the subsequent circuit, for instance $1/2$ for squarings with injected error at most $1/4$.

Our configuration uses eight EvalMod levels with scaling factor 2^{28} , the smallest tested scaling factor at which every run leaves the guard below $2^{-1.5}$ after bootstrapping. The measured guard is at most $2^{-1.79}$, against $2^{-2.13}$ for one shaped rounding. Raising the EvalMod scaling factor from 2^{26} to 2^{28} costs 16 bits, 14 fewer than a separate 30-bit mask, and saves a plaintext product, leaving room for 21 levels instead of 20.

5.4 Correctness and Modulus Budget

The transform identities establish exact recovery, while the preceding error bounds determine the working precision and the output guard bound.

Proposition 3 (Fiber-aware bootstrapping). *Assume $R \mid k$ and that the desired coefficient message lies in the centered reduction interval modulo q_b . In exact arithmetic, replacing EvalMod by exact modular reduction makes the pipeline recover the k input values in the replicated layout with zero guards. Each transform uses at most k diagonals, and the bottom and top traces use $\log_2 d$ and $\log_2(N/(2k))$ rotations, respectively.*

Proof. The bottom trace preserves the replicated data, and Lemma 5 makes the projection of the forward transform equal to sparse SlotsToCoeffs. Writing $\mathcal{T}$ for the normalized top trace, the lifted phase satisfies

$$\mathcal{T}(\mu + q_b I) = \mathcal{T}(\mu) + q_b \mathcal{T}(I),$$

where $\mathcal{T}(I)$ is integral because $\mathcal{T}$ discards coefficient positions. CoeffsToSlots exposes these coefficients, so modular reduction removes $q_b \mathcal{T}(I)$ and recovers the message, with zero guard rows by Lemma 7. The transform lemmas give the diagonal counts, and successive doubling gives the trace counts. $\square$

When evaluating this bootstrapping algorithm, the lifted coefficients must lie in the EvalMod approximation domain and all intermediate phases must satisfy the no-wrap bounds. The data error combines the incoming error, the final ordinary SlotsToCoeffs rounding, the wide-stage residuals of Lemma 8, and the approximation and arithmetic errors of EvalMod propagated using Section 4.1. The output guard condition is given in Section 5.3. Separately from guard escape, the standard uniform-carry model for sparse-secret encapsulation [BTH22] estimates EvalMod range failure at approximately $2^{-139.7}$ for weight 32, interval radius 16 and $N = 2^{15}$, using a union bound over all N coefficients.

The modulus budget must cover the transform and EvalMod rescalings while leaving enough capacity for the subsequent leveled computation. The largest ciphertext and evaluation-key moduli, including temporary extensions, must satisfy the security limits, with the encapsulation key checked separately under its sparse-secret parameters.

6 Instantiation and Implementation

We instantiate our scheme in poulpy [BDM⁺26], and use the bivariate representation [BCG⁺24] with 52-bit limbs and power-of-two moduli, allowing bit-granular rescaling. We choose the shaping unit to meet both the leveled and bootstrapping error bounds, then use its subring structure to reduce encoding and rounding costs.

6.1 Choosing the Shaping Unit

We need strong suppression on the data class without guard amplification that would force a larger leveled scaling factor or leave unstable guards after bootstrapping. At a given suppression, the norm bound of Section 4.4 favors a flat guard profile. The logarithmic embedding turns the search for such a unit into a lattice problem. With $\omega = e^{i\pi/(2R)}$, associate to a unit U its logarithmic profile

$$\ell(U) = (\log_2 |U(\omega^{5^j})|)_{j=0}^{R-1}.$$

Products of units add their profiles, and the norm identity gives $\sum_j \ell_j(U) = 0$. Dirichlet's unit theorem [CDPR16] states that these profiles form a lattice Λ of rank $R - 1$ spanning the hyperplane $H = \{\mathbf{x} \in \mathbb{R}^R : \sum_j x_j = 0\}$. To obtain T bits of suppression with a flat guard profile, we therefore seek a lattice point close to

$$\mathbf{t}_T = \left(-T, \frac{T}{R-1}, \dots, \frac{T}{R-1}\right),$$

where distance in the maximum norm bounds the deviation of every class from its target, measured in bits.

Proposition 4 (Achievable profiles). *Let $\theta_\Lambda = \sup_{\mathbf{x} \in H} \inf_{\mathbf{v} \in \Lambda} \|\mathbf{x} - \mathbf{v}\|_\infty$ be the covering radius of Λ in H . For every $T > 0$, there is a unit U such that*

$$|U(\omega)| \leq 2^{-T+\theta_\Lambda}, \quad \max_{1 \leq j < R} |U(\omega^{5^j})| \leq 2^{T/(R-1)+\theta_\Lambda}.$$

Proof. The target $\mathbf{t}_T$ belongs to H , so there is a lattice point $\ell(U)$ within θ_Λ of it in every coordinate. The data coordinate is therefore at most $-T + \theta_\Lambda$, and every guard coordinate is at most $T/(R-1) + \theta_\Lambda$. Exponentiating gives both bounds. $\square$

For the search, we use cyclotomic units generated by $(1 - Y^a)/(1 - Y)$ for odd a with $1 < a < 2R$, whose logarithmic profiles span a sublattice of finite index in Λ [CDPR16]. We reduce a basis of this sublattice with LLL [LLL82], apply Babai's nearest-plane algorithm to the target $\mathbf{t}_T$, and refine the resulting integer combination by a local search. The same integer combination of the generators, interpreted multiplicatively, gives the candidate unit, which we denote by $U_{2R,T}$ for ring degree $2R$ and target suppression T bits. Its actual suppression

$\log_2(1/m_S)$ may differ from T , and the search does not need to find a closest lattice point.

Proposition 4 leaves the covering radius unspecified, so it does not give a numerical parameter choice. A gap to the norm bound does not establish whether a better unit exists or whether the lattice geometry prevents a flatter profile at that gain.

Choosing R . A larger R improves the norm bound, but does not necessarily improve the units found by the search, since the covering radius also changes. It increases rounding costs, limits k to $N/(2R)$ and requires padding batches smaller than R , so we choose the smallest value meeting both the leveled and bootstrapping requirements.

For a negligible evaluation-key noise contribution, at the threshold $\Delta = 4ME$ a level injects data error $m_S/(4M)$, so a circuit whose propagated rounding contribution is at most L times this error needs

$$\log_2(1/m_S) \geq b_{\text{prec}} + \log_2 L - 2 - \log_2 M$$

to keep that contribution below $2^{-b_{\text{prec}}}$. For $b_{\text{prec}} = 19$, $L = 16$ and $\log_2 M \approx 3.4$, about 18 bits of suppression suffice.

Bootstrapping is more demanding because its rounding errors undergo the amplification described in Section 5.3. The $R = 8$ unit $U_{16,20}$ with 19.3 bits of suppression gives only about 10 bits through bootstrapping in our measurements. The better unit $U_{16,40}$ used during bootstrapping gives a guard band exceeding one at $L = 16$. We use $R = 16$ and $U_{32,40}$, with 39.7 bits of suppression and $\log_2 M \approx 4.7$, throughout the computation. Its measured stability edge at $N = 2^{15}$ is 19 bits, and our levels use 20 bits, with larger margins governed by the failure-probability analysis of Section 4.3.

6.2 Encoding and Decoding in the Subring

Replication makes full-ring encoding unnecessary: the intended plaintext lies in a subring, and shaped rounding preserves it.

Proposition 5 (Subring encoding). *Let m encode $z \in \mathbb{C}^k$ in the replicated layout with zero guards, and let $d = N/(2kR)$. For $\Delta > 0$ and an integer $r \geq 2$,*

$$\text{Round}_{g,r}(\lfloor r\Delta m \rfloor) \in \mathbb{Z}[X^d],$$

with at most $2kR$ nonzero coefficients. Encoding uses a canonical transform for kR complex slots and $O(kR^2)$ coefficient operations for shaped rounding.

Proof. The slot period gives $m \in \mathbb{R}[X^d]$, and $Y = (X^d)^k$ gives $g, g^{-1} \in \mathbb{Z}[X^d]$. Quantization and shaped rounding preserve this subring by the argument of Proposition 2. Its degree is $2kR$, and each of the two products in the rounding uses at most $2R$ shifts, giving the stated costs. $\square$

This gives the same plaintext as full-ring encoding in exact arithmetic. At $N = 2^{15}$, encoding 16 and 64 values takes 148 and 182 microseconds, respectively, against 971 and 978 microseconds in the full ring.

Decoding also avoids a full-ring transform and can improve precision by averaging the replicas instead of reading one as in Section 3.3. We first project the decrypted phase

$$\mu_{\text{av}} = d^{-1} \text{Tr}_{K/K_d}(\mu) = \sum_{j=0}^{2kR-1} \mu_{dj} X^{dj},$$

then apply the smaller transform and return only the data slots. This plaintext operation averages the replicas without keyswitching, so its error is at most their largest error. Independent, centered errors of equal variance gain $\frac{1}{2} \log_2 d$ bits, while periodic errors are unchanged, and independence is not assumed for bootstrapping errors. For $k = 16$, averaging raises measured minimum precision from 48.75 to 51.0 bits, a gain of 2.3 bits against 3 predicted, while $k = 64$ gains 1.2 bits against 2 predicted and $k = 1024$ gains nothing because $d = 1$. Subring decoding at $k = 16$ takes 2 microseconds against 134 in the full ring.

6.3 The Shaped-Rounding Pass

Shaped rounding only needs to process the dropped bits, since the retained part contributes an integer quotient. The sparse support of g and its modular inverse then lets us compute the correction with word arithmetic. For a drop of $b \geq 1$ bits, write $c = c_{\text{low}} + 2^b c_{\text{high}}$ with $0 \leq (c_{\text{low}})_j < 2^b$ and compute

$$\begin{aligned} h &= [(g^{-1} \bmod 2^b) c_{\text{low}}]_{2^b}, \\ \kappa &= (c_{\text{low}} - gh)/2^b, \\ \text{Round}_{g,2^b}(c) &= c_{\text{high}} + \kappa. \end{aligned}$$

The first product is modular, while the carry numerator is computed over the integers and is divisible by 2^b . Since g and its precomputed inverse modulo 2^b each have at most $2R$ terms, the pass uses at most $4RN$ word multiply-adds on the dropped part, followed by the normalization.

Lemma 9 (Word-size exactness). *With $\|g\|_1 = \sum_j |g_j|$,*

$$-\frac{\|g\|_1}{2} \leq \kappa_j < 1 + \frac{\|g\|_1}{2}, \quad |(c_{\text{low}} - gh)_j| < 2^{b-1}(2 + \|g\|_1).$$

The numerator fits a signed 64-bit word if $b \leq 64 - \lceil \log_2(2 + \|g\|_1) \rceil$.

Proof. Since $\|h\|_\infty \leq 2^{b-1}$, we have $\|gh\|_\infty \leq \|g\|_1 2^{b-1}$. Combining this with the range of c_{low} gives both bounds, and the condition on b makes the strict numerator bound at most 2^{63} . $\square$

Our unit has $\|g\|_1 = 45$, allowing passes of up to 58 bits. Larger drops can be split while retaining shaping, since

$$\text{Round}_{g,r_2}(\text{Round}_{g,r_1}(c)) = \frac{c - g(h_1 + r_1 h_2)}{r_1 r_2}.$$

Finite Shaping Depth. To limit the work to one shaped pass, our implementation uses ordinary rounding for the first part of a large drop. The shaped division suppresses this unshaped error, so the pass must be deep enough to keep it below the desired data error.

Lemma 10 (Shaping depth). *Let $r = r_1 r_2$, with integers $r_1 \geq 1$ and $r_2 \geq 2$ and with g invertible modulo r_2 , and set $a' = \lfloor a/r_1 \rfloor$ for $a \in \mathcal{R}$. There are $h, u \in \mathcal{R}$ such that*

$$\begin{aligned} \text{Round}_{g,r_2}(a') &= \frac{a}{r_1 r_2} - g \frac{h}{r_2} - \frac{u}{r_1 r_2}, \\ \|h/r_2\|_\infty &\leq \frac{1}{2}, \quad \|u/(r_1 r_2)\|_\infty \leq \frac{1}{2r_2}. \end{aligned}$$

Proof. Write $a = r_1 a' + u$, where $\|u\|_\infty \leq r_1/2$, and put $h = [(g^{-1} \bmod r_2)a']_{r_2}$. Expanding $\text{Round}_{g,r_2}(a') = (a' - gh)/r_2$ gives the identity and both coefficient bounds. $\square$

We call $\lambda = \log_2 r_2$ the shaping depth. Applied to both ciphertext components, the lemma gives a shaped phase remainder plus an unshaped remainder suppressed by $2^{-\lambda}$. If E bounds the aggregate shaped remainder and the unshaped remainder after multiplication by 2^λ , a level's data error is at most

$$\delta \leq \frac{(m_S + 2^{-\lambda})E + E_{\text{key}}}{\Delta}.$$

Once $\lambda \geq \log_2(1/m_S)$, the unshaped contribution is no larger than the shaped one, and a deeper pass approaches the fully shaped error floor of Section 4.1.

Extra working bits come from modulus extension: for a product with scaling factor Δ^2 , extension by 2^w permits division by $2^w \Delta$ while returning to the original scaling factor Δ , provided the extended modulus is allowed. With w extra bits and a maximum pass size ℓ_{pass} , the shaping depth is $\lambda = \min(\log_2 \Delta + w, \ell_{\text{pass}})$. With scaling factor 2^{20} , changing (w, ℓ_{pass}) from $(8, 32)$ to $(24, 48)$ reduces the measured error added per product from $2^{-34.7}$ to $2^{-46.3}$, close to the fully shaped estimate, for 4.7 percent more time. Deeper shaping is therefore useful for preserving a high-precision input through leveled computations.

6.4 Representation and Evaluation Keys

Apart from replacing rounding by the pass above, the implementation uses poulpy's existing arithmetic and ordinary evaluation keys everywhere. These keys suffice because, in the bivariate representation, the contribution from their sampled noise is suppressed when the ciphertext modulus is narrower than the key modulus. An earlier sweep retained at least 21.2 bits after bootstrapping followed by 16 levels with every tested auxiliary modulus, down to 4 bits. Shaping all bootstrapping keys instead lost 3 to 4 bits because the rotation keys act in the coefficient domain, while shaping only the relinearization key made no

difference. Grouping two limbs per digit reduces the number of key components but requires a larger auxiliary modulus: the parameter sweep selects $\log_2 p = 53$ and 67 for one- and two-limb digits, respectively, subject to the evaluation-key modulus limit.

With RNS-CKKS [CHK⁺19], ordinary keys and an auxiliary modulus comparable to the largest digit can give an evaluation-key noise contribution $p^{-1} \sum_i c_{1,i} e_i$ that dominates the shaped data error. Shaped slot-domain keys or a larger auxiliary modulus control this term, which is already negligible in our selected bivariate configurations.

Remark 1 (RNS instantiation). Compute $h = [(g^{-1} \bmod r)c]_r$, form gh in the retained residues and divide $c - gh$ by the dropped factor r . A base conversion returning $h + ru$ changes the result by $-gu$, preserving shaping with a larger remainder bound.

7 Experimental Results

7.1 Experimental Setup

We implement our scheme in `poulpy v0.9.0` [BDM⁺26] and run the experiments on a MacBook equipped with an Apple M5 Pro and 64 GiB of RAM, using one thread or all 18 cores. The main grid uses $k \in \{16, 64, 256, 1024\}$ values and restores $L \in \{1, 2, 4, 8, 16, 21\}$ levels, with five seeds and inputs sampled uniformly on the unit circle or in the unit disk. Following `poulpy`’s `precision_stats`, minimum precision is minus the base-two logarithm of the largest absolute error over the real and imaginary parts of the data slots, and average precision uses their mean absolute error. Times are medians over the five seeds, and minimum and average precision are measured after L products by fresh operands, taking the worst and mean values over seeds and input families, respectively.

Our parameters are $N = 2^{15}$, $R = 16$, the unit $U_{32,40}$ of Section 6.1, and scaling factor $\Delta = 2^{20}$. Batches with fewer than 16 values are padded with zeros to meet the transform condition $R \mid k$, giving the computation of the $k = 16$ rows. We compare against `poulpy`’s sparse CKKS bootstrapping with 36-bit levels using factored transforms for $k \geq 256$ and increasing N to 2^{16} when needed. Both schemes use sparse ternary secrets of Hamming weight $h_{\text{wt}} = 1024$. Ephemeral encapsulation secrets use Hamming weight 32. For 128-bit security, we use the modulus caps in the `sparse-key-estimate` tables [Dum26], which include Ogilvie’s attack on sparse secrets [Ogi26]. At $N = 2^{15}$, the caps on the maximum moduli $\log_2(qp)$ are 851 and 162 bits for weights 1024 and 32, respectively, and at $N = 2^{16}$ they are 1707 and 345 bits.

Figure 3 gives our modulus chain. `SlotsToCoeffs` consumes 26 bits, taking the input modulus from 2^{69} to $q_b = 2^{43}$ with coefficient-message scaling factor $\Delta_b = 2^{30}$, hence $q_b/\Delta_b = 2^{13}$. After `ModUp`, `CoeffsToSlots` uses 34-bit diagonals, and `EvalMod` uses a degree-30 cosine approximation on the interval of radius 16, three double-angle steps following Han and Ki [HK20], scaling factor 2^{28} and 26-bit constants. The guard reset is folded into these constants, and the last

squaring is left unrescaled, so the final division removes $30 + 28 - 20 = 38$ bits. Including six extra bits in the CoeffsToSlots scaling factor, bootstrapping consumes $(34 + 6) + 196 + 38 = 274$ bits after ModUp, giving $\log_2 q_{\text{boot}} = 343 + 20L$.

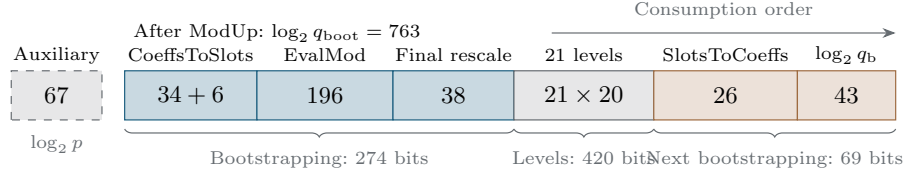


Fig. 3. Modulus allocation for $L = 21$, in bits, with the auxiliary keyswitching modulus on the left and ciphertext blocks consumed from left to right after ModUp. The CoeffsToSlots budget includes six extra bits in its scaling factor, and the final 69 bits support the next SlotsToCoeffs and ModUp. For other depths, replace 420 by $20L$ and select p as below. Block widths are schematic.

Evaluation keys cover eight additional working bits, rounded up to 52-bit limbs, giving $\log_2 q_{\text{key}} = 52 \lceil (\log_2 q_{\text{boot}} + 8) / 52 \rceil$ before the auxiliary modulus. We use $\log_2 p = 53$ for $L \leq 8$ and 67 for $L \geq 16$, with one- and two-limb digits, respectively. At $L = 21$, this gives $\log_2(q_{\text{key}}p) = 780 + 67 = 847 < 851$, while the key under the ephemeral secret uses $52 + 67 = 119 < 162$ bits.

7.2 Bootstrapping Latency and Available Depth

Tables 3 and 4 compare our bootstrapping with the fastest measured sparse CKKS configuration meeting the security caps and retaining at least 19 bits of minimum precision after L products by fresh operands.

The smaller modulus per level lets our scheme retain $N = 2^{15}$ when sparse CKKS needs $N = 2^{16}$, reducing both latency and memory, with the speedups summarized in Table 5. The baseline is faster at $k = 1024$ with one or two levels because it factors its transforms, while our fiber-aware transforms still use k diagonals by Lemmas 5 and 7. Profiling bootstrapping on one thread at $k = 1024$ attributes 38 percent of samples to products of baby rotations by diagonals. In an additional 200 bootstrapping runs over five key sets at $k = 16$ and $L = 16$, minimum precision after bootstrapping was 21.87 bits, with guards at most $2^{-1.79}$ after bootstrapping and $2^{-1.44}$ over the subsequent squarings.

7.3 Leveled Precision and Stability

Figure 4 measures 16 products by fresh operands without bootstrapping, using $N = 2^{15}$, $k = R = 16$, $U_{32,40}$, five seeds and both input families. With scaling factor 2^{20} , our scheme retains 44.7 bits of minimum precision against 5.1 bits for

Table 3. Bootstrapping measurements for 16 and 64 values, with 20-bit levels for our scheme and 36-bit levels for sparse CKKS. Precision is measured after L products by fresh operands. The maximum evaluation-key modulus is given in bits, and memory is peak resident GiB of the single-thread process.

L	Scheme	Parameters		Precision (bits)		Time (s)		Memory
		$\log_2 N$	Max. modulus $\log_2(q_{\text{key}}p)$	Min.	Avg.	1 thread	18 cores	(GiB)
$k = 16$ values								
1	Ours	15	469	22.8	24.8	0.548	0.171	3.2
	Sparse	15	706	21.2	23.8	0.840	0.206	5.9
16	Ours	15	743	23.0	24.8	1.044	0.273	4.6
	Sparse	16	1300	19.7	23.3	3.754	0.653	20.1
21	Ours	15	847	22.8	24.9	1.201	0.284	5.8
	Sparse	16	1628	19.7	23.2	4.438	0.760	16.4
$k = 64$ values								
16	Ours	15	743	22.8	24.8	1.232	0.273	5.7
	Sparse	16	1300	19.2	21.8	4.388	0.770	30.7

Table 4. Bootstrapping measurements for larger batches and the full-packing preset, with columns and units as in Table 3. The preset uses 35-bit levels and reports poulpy’s own precision and timings under its own protocol, secret and security parameters.

L	Scheme	Parameters		Precision (bits)		Time (s)		Memory
		$\log_2 N$	Max. modulus $\log_2(q_{\text{key}}p)$	Min.	Avg.	1 thread	18 cores	(GiB)
$k = 256$ values								
16	Ours	15	743	22.2	24.6	1.651	0.336	8.3
	Sparse	16	1628	20.0	23.5	5.233	0.852	30.4
$k = 1024$ values								
1	Ours	15	469	21.6	24.1	1.768	0.405	11.0
	Sparse	15	833	20.8	23.2	1.576	0.322	12.9
16	Ours	15	743	21.5	24.1	2.788	0.546	15.1
	Sparse	16	1628	19.9	23.5	5.865	0.940	40.7
Poulpy n16_d35_k720_p19_s2c, $k = 32768$								
16	CKKS	16	1606	18.9	–	5.490	0.780	–

CKKS. The precision plateau on the left comes from double-precision decoding, while double-double decoding resolves the smaller per-level errors on the right.

For the model in Table 6, $\hat{E}_{\text{guard}}$ is the largest injected guard error measured at $b_{\Delta} = 28$, and $\hat{E}_{\text{data}}$ is the injected data error at the measured edge, both

Table 5. Bootstrapping speedups over the full grid, expressed as baseline time divided by our time. The two baseline points at $L = 21$, $k \in \{256, 1024\}$ are excluded because memory compression distorts their timings.

Comparison	1 thread	18 cores
Both $N = 2^{15}$, $k \leq 256$	1.21–1.61	1.05–1.40
Both $N = 2^{15}$, $k = 1024$	0.89–0.92	0.80
Baseline $N = 2^{16}$, ours $N = 2^{15}$	1.94–3.79	1.51–2.82

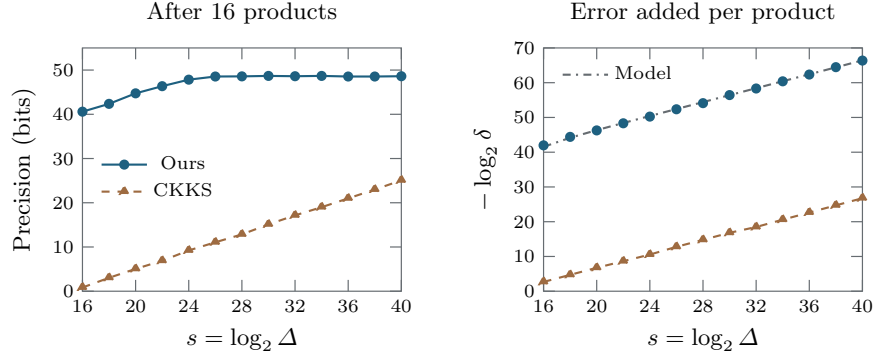


Fig. 4. Minimum data precision after 16 products (left) and $-\log_2 \delta$ for the largest error added by one product (right), over all levels, seeds and input families. Writing $b_\Delta = \log_2 \Delta$, the model is $\delta = 2^{13.3-b_\Delta}(2^{-\lambda} + m_S)$ with shaping depth $\lambda = \min(b_\Delta + 24, 48)$, as in Lemma 10.

in phase units. These measured totals include rounding errors, so substituting $\hat{E}_{\text{guard}}$ for ME and $\hat{E}_{\text{data}}$ for E_{key} gives an empirical prediction, not an isolated measurement of the evaluation-key noise contribution. The measured edges never exceed this empirical prediction and lie 0.5 to 1.3 bits below it. One bit below each measured edge, the largest guard exceeds one within 16 squarings for all ten unit and ring-dimension configurations.

Activations. We compose four evaluations of a degree-seven interpolant of $\tanh(2x)/\tanh(2)$ at Chebyshev nodes, whose slope at zero is about 2.06. We use real data in $[-1, 1]$, scaling factor 2^{20} , $k = 16$ and five seeds. Without help, the largest guard grows from $2^{-2.5}$ after the first activation to $2^{3.3}$ after the fourth. Its slope exceeds one, so the absolute-coefficient condition of Section 4.2 gives no invariant-disk guarantee. Realification reduces the final magnitude to $2^{1.3}$, while a reset before each activation keeps it at most $2^{-0.9}$ for 80 additional modulus bits over the four activations. A product by a fresh operand after each activation also keeps it at most $2^{-0.9}$. With scaling factor 2^{24} , realified guards still grow by about one bit per activation, from $2^{-5.8}$ to $2^{-2.9}$, since zero is a repelling fixed point of the activation. The data keep more than 41.9 bits of minimum precision in every variant.

Table 6. Measured stability edges over 16 squarings, with $k = 16$, five seeds and both input families. The edge is the smallest tested integer $b_\Delta = \log_2 \Delta$ for which all guards stay below one with that scaling factor and every larger tested one. The tested exponents range from 8 to 28. The empirical model evaluates $\log_2(4(\hat{E}_{\text{guard}} + \hat{E}_{\text{data}}))$ using the measured one-step errors defined below.

$\log_2 N$	Unit	$-\log_2 m_S$	$\log_2 M$	Model	Measured
15	$U_{16,20}$	19.3	3.40	19.22	18
15	$U_{16,40}$	39.3	7.04	22.78	22
15	$U_{16,60}$	62.0	9.90	25.23	24
15	$U_{32,20}$	21.8	4.26	20.02	19
15	$U_{32,40}$	39.7	4.70	20.06	19
15	$U_{32,60}$	67.1	5.69	21.29	20
15	$U_{64,20}$	24.0	3.85	19.30	18
15	$U_{64,40}$	40.7	4.09	19.65	19
15	$U_{64,60}$	60.5	5.05	20.50	20
16	$U_{32,40}$	39.7	4.70	20.88	20

7.4 Comparison with PaCo

We benchmark poulpy’s encapsulated PaCo [CS25] under the same protocol, tuning its scaling factors and transform schedules to retain at least 19 bits of minimum precision. The application secret has weight 1024 and its keys satisfy the same modulus caps, but PaCo’s structured binary secret of weight $h_P = 64$ requires a separate security assumption, outside those tables.

Table 7. PaCo comparison, with each paired entry giving PaCo / ours. Precision is the minimum after L products by fresh operands, and t_1, t_{18} are bootstrapping times in seconds. The ring dimension and maximum evaluation-key modulus (in bits) refer to PaCo, while our scheme uses $N = 2^{15}$ throughout. PaCo branches run sequentially, with parallelism inside each branch on 18 cores.

k	L	$\log_2 N$	Max. modulus $\log_2(q_{\text{key}p})$	Precision	t_1	t_{18}
16	1	15	691	20.1 / 22.8	0.723 / 0.548	0.168 / 0.171
16	4	15	795	20.2 / 22.8	0.916 / 0.632	0.205 / 0.191
16	16	16	1316	19.9 / 23.0	3.718 / 1.044	0.685 / 0.273
64	1	15	691	19.4 / 22.8	1.198 / 0.662	0.277 / 0.183
64	4	15	795	19.5 / 22.8	1.547 / 0.747	0.361 / 0.211
256	1	15	743	22.4 / 22.3	4.881 / 1.015	1.147 / 0.246
1024	1	15	743	21.4 / 21.6	19.504 / 1.768	4.711 / 0.405

PaCo matches our all-core latency for 16 values at shallow depth, but its 38- to 42-bit levels leave room for only four to six levels at $N = 2^{15}$, against 21 for our scheme. Each branch recovers at most $N/(4h_P)$ real coefficients, so at $N = 2^{15}$ and $h_P = 64$, batches of 256 and 1024 complex values require four and sixteen branches, respectively. Running the branches concurrently could lower the all-core latency at 256 and 1024 values, which we did not measure.

Acknowledgments. We thank Aymen Boudguiga, Pierre-Emmanuel Clet and Olive Chakraborty for discussion and feedback on an earlier version of this work.

Generative AI Disclosure The authors used OpenAI ChatGPT Astra 6 and Anthropic Claude Opus 5.5 to improve the English and clarity of the manuscript, assist in checking mathematical proofs, and support code development for the experimental section. All proofs, code, and experimental results were reviewed and verified by the authors, who take full responsibility for the content of this work.

References

- AKP25. Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using CKKS. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 304–337, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.
- BCG⁺24. Mariya Georgieva Belorgey, Sergiu Carpov, Nicolas Gama, Sandra Guasch, and Dimitar Jetchev. Revisiting key decomposition techniques for FHE: Simpler, faster and more generic. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part I*, volume 15484 of *Lecture Notes in Computer Science*, pages 176–207, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.
- BDM⁺26. Jean-Philippe Bossuat, Jules Dumezy, Rasoul Akhavan Mahdavi, Janmajaya Mall, Cedoor, Luis Ruiz-Lopez, and Christian Mouchet. Poulpy v0.9.0. Online: <https://github.com/poulpy-fhe/poulpy>, 2026.
- BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery.
- BKSS24. Youngjin Bae, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Bootstrapping small integers with CKKS. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part I*, volume 15484 of *Lecture Notes in Computer Science*, pages 330–360, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.
- Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.

- BTH22. Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 2022: 20th International Conference on Applied Cryptography and Network Security*, volume 13269 of *Lecture Notes in Computer Science*, pages 521–541, Rome, Italy, June 20–23, 2022. Springer, Cham, Switzerland.
- BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.
- CDPR16. Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 559–585, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.
- CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.
- CHK⁺18. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 360–384, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.
- CHK⁺19. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J.: Jacobson, Jr., editors, *SAC 2018: 25th Annual International Workshop on Selected Areas in Cryptography*, volume 11349 of *Lecture Notes in Computer Science*, pages 347–368, Calgary, AB, Canada, August 15–17, 2019. Springer, Cham, Switzerland.
- CK25. Jean-Sébastien Coron and Robin Köstler. Low-latency bootstrapping for CKKS using roots of unity. Cryptology ePrint Archive, Report 2025/651, 2025.
- CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.
- CLPX18. Hao Chen, Kim Laine, Rachel Player, and Yuhou Xia. High-precision arithmetic in homomorphic encryption. In Nigel P. Smart, editor, *Topics in Cryptology – CT-RSA 2018*, volume 10808 of *Lecture Notes in Computer Science*, pages 116–136, San Francisco, CA, USA, April 16–20, 2018. Springer, Cham, Switzerland.
- CS25. Jean-Sébastien Coron and Tim Seuré. PaCo: Bootstrapping for CKKS via partial CoeffToSlot. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025, Part VII*, volume 16251 of *Lecture Notes in Computer Science*, pages 36–67, Melbourne, VIC, Australia, December 8–12, 2025. Springer, Singapore, Singapore.

- DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.
- Dum26. Jules Dumezy. Sparse Key Estimate. Online: <https://github.com/jdumezy/sparse-key-estimate>, 2026.
- FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.
- Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.
- GHS12a. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.
- GHS12b. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.
- GP25. Antonio Guimarães and Hilder V. L. Pereira. Fast amortized bootstrapping with small keys and polynomial noise overhead. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, pages 2967–2981, Taipei, Taiwan, October 13–17, 2025. ACM Press.
- GPvL23. Antonio Guimarães, Hilder V. L. Pereira, and Barry van Leeuwen. Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part VI*, volume 14443 of *Lecture Notes in Computer Science*, pages 3–35, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.
- GV25. Robin Geelen and Frederik Vercauteren. Fully homomorphic encryption for cyclotomic prime moduli. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part III*, volume 15603 of *Lecture Notes in Computer Science*, pages 366–397, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.
- HK20. KyooHyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, volume 12006 of *Lecture Notes in Computer Science*, pages 364–390, San Francisco, CA, USA, February 24–28, 2020. Springer, Cham, Switzerland.
- HS14. Shai Halevi and Victor Shoup. Algorithms in HELib. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2014. Springer Berlin Heidelberg, Germany.

- KS19. Duhyeong Kim and Yongsoo Song. Approximate homomorphic encryption over the conjugate-invariant ring. In Kwangsu Lee, editor, *ICISC 18: 21st International Conference on Information Security and Cryptology*, volume 11396 of *Lecture Notes in Computer Science*, pages 85–102, Seoul, Korea, November 28–30, 2019. Springer, Cham, Switzerland.
- LLL82. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
- LM21. Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 648–677, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.
- LMSS22. Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 560–589, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.
- LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.
- LWZ⁺26. Zhihao Li, Hongyu Wang, Yuan Zhao, Lichun Li, Zhiwei Wang, Jiaxing He, Changzheng Wei, Ying Yan, and Lifeng Guo. BatchBoot: Fast batched bootstrapping for TFHE scheme and practical applications. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 21–40, Baltimore, MD, August 2026. USENIX Association.
- Ogi26. Tabitha Ogilvie. On the concrete hardness gap between MLWE and LWE. Cryptology ePrint Archive, Paper 2026/279, 2026.
- PZZ26. Chris Peikert, Doron Zarchy, and Guy Zyskind. High-precision exact FHE made simple, general, and fast. In Nadia Heninger and Mike Rosulek, editors, *CRYPTO 2026, Part II*, volume 16801 of *LNCS*, pages 525–557. Springer, Cham, 2026.
- TBC⁺25. Daphné Trama, Aymen Boudguiga, Pierre-Emmanuel Clet, Renaud Sirdey, and Nicolas Ye. Designing a general-purpose 8-bit (T)FHE processor abstraction. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2):535–578, 2025.
- Zhe26. Xiaopeng Zheng. Faster CoeffToSlot and SlotToCoeff for sparsely packed ciphertexts with application to CKKS bootstrapping. Cryptology ePrint Archive, Paper 2026/1023, 2026. ASIACRYPT 2026.

A Conversions to and from CKKS

Both schemes use the phase $[c_0 + c_1 s]_q$ and share the secret and evaluation keys, so conversion only changes the layout and scaling factor. Shaped keys remain valid CKKS keys, with their errors bounded on every slot used by the computation. We use the replicated layout with $d = N/(2kR)$ and the CKKS layout with intended values $z_{t \bmod k}$, allowing different errors among replicas. Write Δ_c ,

Δ and Δ_d for the CKKS, leveled and diagonal scaling factors, respectively. All division factors must divide the corresponding moduli, and temporary cipher-text and evaluation-key moduli must meet the security limits. Both conversions consume $\log_2 \Delta_d$ bits of *headroom*, defined as $\log_2(q/\Delta)$.

A.1 From CKKS to the Replicated Layout

Let A write $z_{\iota(t)}$ on S and zero on the guards. Its k diagonals from Lemma 7 have entries zero or one and are encoded with shaping at scaling factor Δ_d . Let U and U' bound their shaped remainders before multiplication by g and their unshaped remainders, respectively, in canonical norm. For a pre-rotated diagonal followed by giant rotation j , use $\tau_{5^{-j}}(g)$ when encoding, since $\tau_{5^j}(\tau_{5^{-j}}(g)\rho) = g\tau_{5^j}(\rho)$.

Starting from modulus q and scaling factor Δ_c , choose an integer $w \geq 0$ and apply the following operations.

1. Extend the modulus and scaling factor by 2^w .
2. Evaluate A with ordinary internal keyswitching, then rescale by Δ_d with ordinary rounding.
3. Apply shaped rescaling by $r = 2^w \Delta_c / \Delta$, omitting this operation when $r = 1$.

Proposition 6 (Conversion from CKKS). *Suppose $|x_t - z_{t \bmod k}| \leq \varepsilon$ on every input slot and $\|z\|_\infty \leq D$. Let E_{int} bound internal rounding and evaluation-key noise propagated through the encoded diagonals and ordinary rescale to the phase before the final division, excluding diagonal quantization acting on the input. Let E_{rnd} bound the final shaped remainder, with $E_{\text{rnd}} = 0$ when $r = 1$, and put $\theta = E_{\text{int}}/(2^w \Delta_c) + kU'(D + \varepsilon)/\Delta_d$. If no phase wraps around, the output has modulus $q\Delta/(\Delta_d \Delta_c)$, scaling factor Δ , and satisfies*

$$\begin{aligned} |x'_t - z_{\iota(t)}| &\leq \varepsilon + \theta + m_S \left(\frac{kU(D + \varepsilon)}{\Delta_d} + \frac{E_{\text{rnd}}}{\Delta} \right) & (t \in S), \\ |x'_t| &\leq \theta + M \left(\frac{kU(D + \varepsilon)}{\Delta_d} + \frac{E_{\text{rnd}}}{\Delta} \right) & (t \notin S). \end{aligned}$$

Proof. The exact map reads one input value at each data output and writes zero on the guards. Each diagonal remainder multiplies a value of magnitude at most $D + \varepsilon$, giving the quantization terms after summing over k diagonals. The internal errors contribute $E_{\text{int}}/(2^w \Delta_c)$, and the final division adds at most $|\sigma_t(g)|E_{\text{rnd}}/\Delta$. The modulus becomes $2^w q/(\Delta_d r)$, giving the stated modulus and headroom cost. $\square$

The extra working bits suppress internal arithmetic errors without consuming headroom, while diagonal quantization requires sufficient encoding precision. The output guards must satisfy the initial condition of the subsequent circuit.

A.2 From the Replicated Layout to CKKS

Assume $R \mid k$, padding smaller batches as in Section 5, and put $h = dR = N/(2k)$. The gather diagonals of Lemma 5 with $A = I$ have nonzero entries R and read only data slots. The normalized full trace onto K_h then returns the period- k layout, averaging the d copies of each value without a separate bottom trace.

Lemma 11 (Trace at a lifted modulus). *Let a ciphertext have phase μ modulo q and scaling factor Δ_0 , with $h \mid N/2$. Reinterpret its components modulo hq and evaluate Tr_{K/K_h} at that modulus. The output phase is $[\text{Tr}_{K/K_h}(\mu) + e_{\text{tr}}]_{hq}$, where e_{tr} collects the propagated keyswitching errors. If this phase does not wrap around, scaling factor $h\Delta_0$ gives the normalized trace plus its error, with unchanged headroom.*

Proof. Write $c_0 + c_1 s = \mu + qI$ with $I \in \mathcal{R}$. Since $\text{Tr}_{K/K_h}(I) \in h\mathcal{R}$, the lifted multiple vanishes modulo hq after the full trace. The modulus and scaling factor both increase by h . $\square$

Choose an integer $w \geq 0$ with $\Delta_c = 2^w h \Delta$, and encode the gather diagonals with ordinary rounding at scaling factor Δ_d , with remainder bound U in canonical norm.

1. Extend the modulus and scaling factor by 2^w .
2. Evaluate the gather with ordinary keyswitching, then rescale by Δ_d with ordinary rounding, giving modulus $2^w q / \Delta_d$ and scaling factor $2^w \Delta$.
3. Lift the modulus by h and evaluate the full trace with ordinary keyswitching, interpreting the output at scaling factor Δ_c .

Proposition 7 (Conversion to CKKS). *Suppose the input data error is at most ε , $\|\mathbf{z}\|_\infty \leq D$, and the guard magnitudes are at most G . Put $B = \max(D + \varepsilon, G)$, and let E_{int} bound rounding and evaluation-key noise propagated through the encoded diagonals, rescaling and unnormalized trace to the final phase, excluding diagonal quantization acting on the input. If the gather stages and final traced phase do not wrap around, the output has modulus $2^w h q / \Delta_d$, scaling factor Δ_c , and satisfies*

$$|x'_t - z_{t \bmod k}| \leq \varepsilon + \frac{kUB}{\Delta_d} + \frac{E_{\text{int}}}{\Delta_c} \quad (t \in \mathbb{Z}_{N/2}).$$

Proof. For output class c modulo k , the gather selects one position in each fiber with weight R , reading all d copies of value z_c across the repetitions. The full trace divides their sum by $h = dR$, so the exact output is their average, with error at most ε . Each diagonal remainder contributes at most UB/Δ_d , and averaging does not increase this bound. The remaining error is bounded by E_{int}/Δ_c . Extension and the lifted trace multiply both modulus and scaling factor by 2^w and h , respectively, while rescaling consumes $\log_2 \Delta_d$ bits of headroom. $\square$

Quantized diagonals can read guards, which accounts for G in the bound, and their exact weights R must be included in the intermediate no-wrap bounds. The error E_{int} includes amplification by the unnormalized trace, whose partial sums may still contain lifted multiples of the modulus. Each conversion uses one transform with at most k diagonals, and the return conversion adds $\log_2 h$ trace rotations by $k, 2k, \dots, (h/2)k$.

B IND-CPA^D Security

Releasing approximate decryptions can expose secret-dependent errors and enable key recovery despite IND-CPA security [LM21]. The usual countermeasure adds flooding noise calibrated to a public bound on the output error, the number of releases and the security target [LMSS22]. Our decryptor releases only the k data values, after selecting or averaging their replicas, and never releases the phase polynomial or guard values.

A leveled operation adds data error at most $((m_S + 2^{-\lambda})E + E_{\text{key}})/\Delta$, where λ is the shaping depth and $2^{-\lambda} = 0$ for fully shaped rounding. When the propagated shaped contribution dominates the output error, the flooding magnitude can be reduced by the same factor m_S at fixed release dimension and security target, about $2^{-39.7}$ for $U_{32,40}$. Unshaped input errors, finite-depth remainders and evaluation-key noise must still be included in the bound.

After bootstrapping, this bound must also cover the unshaped errors of the coefficient-domain stages and transforms, and the EvalMod approximation error, which can depend on the secret through ModUp. The measured precision of about 22 bits is before flooding, and these errors do not benefit from the factor m_S . Guard escape and bootstrapping failures must also fit the security budget, since they can invalidate the output error bound even when guards are not released. The heuristic estimates of Section 4.3 do not by themselves establish IND-CPA^D security, which requires a calibrated release mechanism and error and failure bounds for the supported computations.

C Supplementary Guard Analysis

C.1 Realified Polynomial Evaluations

Realification places the guards near a real interval, with the width of the neighborhood determined by its keyswitching error.

Lemma 12 (Realified polynomial guards). *Let P have real coefficients, $I = [-A, A]$ with $A > 0$, let ε_{R} bound the error added by realification, and let δ_P be as in Section 4.2. If*

$$\sup_{z \in I+B(\varepsilon_{\text{R}})} |P(z)| + \delta_P \leq A,$$

then, starting from $B(A)$, alternating realification with the evaluation of P keeps the guards in $B(A)$ after each evaluation and in $I+B(\varepsilon_{\text{R}})$ after each realification.

Moreover,

$$\sup_{z \in I+B(\varepsilon_R)} |P(z)| \leq \varrho^{\deg P} \max_{x \in I} |P(x)|, \quad \varrho = 1 + \frac{\varepsilon_R}{A} + \sqrt{\frac{2\varepsilon_R}{A} + \frac{\varepsilon_R^2}{A^2}}.$$

Proof. Realification maps $B(A)$ into $I+B(\varepsilon_R)$, and polynomial evaluation maps this set into $P(I+B(\varepsilon_R)) + B(\delta_P) \subseteq B(A)$. The ellipse with foci $\pm A$ and semimajor axis $A+\varepsilon_R$ contains $I+B(\varepsilon_R)$, and its parameter is ϱ , so the Bernstein bound for a polynomial of degree $\deg P$ on this ellipse gives the estimate. $\square$

C.2 Guard-Escape Estimate

We estimate guard escape with the recurrence $v_{j+1} = v_j^2 + \eta_{j+1}$ under the Gaussian model of Section 4.3. A sequence of errors on slot t has Gaussian weight proportional to $\exp(-\sum_j |\eta_j|^2 / \gamma_t(s)^2)$. The radius 1.2 lies beyond the repelling fixed point near one, from which squaring makes the guard grow without further help from the errors. Numerically minimizing $\sum_j |\eta_j|^2$ over paths reaching this radius gives costs $J(b_0)$ of approximately 0.66, 0.62 and 0.52 from initial radii $b_0 = 0, 0.2$ and 0.37 , respectively. These costs give an exponential estimate of the escape probability, not a rigorous tail bound.

The secret must be averaged over its sampling distribution because it determines the noise variance on each slot. For a horizon of L levels, summing the slotwise estimates over guard slots and starting levels gives

$$\hat{P}_{\text{esc}} = L \sum_{t \notin S} \mathbb{E}_{s \leftarrow \chi_{\text{sk}}} \left[\exp \left(-\frac{J(b_0)}{\gamma_t(s)^2} \right) \right],$$

which does not require independence between slots. For Table 2, we model $|\sigma_t(s)|^2$ as exponential with mean $h_{\text{wt}} = 1024$, with $N = 2^{15}$, $L = 21$ and the unit $U_{32,40}$. The average is sensitive to rare secrets with a large slot value where the shaping polynomial is also large, so an estimate for one typical secret can substantially understate it. The estimates rely on uniform independent remainders, Gaussian approximations for the remainders and the secret evaluations, and the escape-path approximation.