# Finite-Field Arithmetic in CKKS

Tim Seuré and Elias Suvanto

University of Luxembourg, Esch-sur-Alzette, Luxembourg
{tim.seure, elias.suvanto}@uni.lu

**Abstract.** We propose a CKKS-based technique for evaluating arithmetic over finite fields $\mathbb{F}_{p^r}$ with small characteristic $p$ under homomorphic encryption. The core of our approach is a pair of complementary ciphertext representations. In the so-called *spectral encoding*, ciphertext addition and multiplication realize addition and multiplication in the field $\mathbb{F}_{p^r}$. In another encoding, *coefficient encoding*, the same operations act as slotwise addition and multiplication in the slot algebra $\mathbb{F}_p^r$. We show that one can switch homomorphically between these encodings at cost linear in $r$, and that $\mathbb{F}_p$-linear maps, such as taking $p$-th powers in $\mathbb{F}_{p^r}$, can be folded into these switches or applied directly in either representation. We complement the construction with theoretical and practical correctness-management techniques. To support unbounded computations, we integrate our framework with existing CKKS bootstrapping techniques and benchmark it against BGV-based implementations of $\mathbb{F}_{p^r}$-arithmetic, a natural baseline for high-throughput finite-field computation. Across the fields we tested, this yields speedups ranging from 1.7× to 178× in amortized multiplication time when bootstrapping is taken into account. The gains are parameter-dependent: roughly speaking, our advantage over BGV increases as the characteristic $p$ becomes smaller and the extension degree $r$ becomes larger.

**Keywords:** CKKS Scheme · Encoding Techniques · Finite-field Arithmetic · Homomorphic Encryption · Lattices · RLWE.

## 1 Introduction

We first review fully homomorphic encryption and the main scheme families, with emphasis on CKKS [16] and its recent discrete variants, and then provide a more detailed overview of our method.

### 1.1 General Overview

**Homomorphic Encryption.** HE (homomorphic encryption) allows computations to be carried out directly on encrypted data, so that sensitive inputs remain protected throughout processing. FHE (fully homomorphic encryption) extends this to the evaluation of arbitrary circuits on ciphertexts, meaning that the computation is performed *homomorphically* without ever decrypting intermediate values. Since [31] introduced bootstrapping in 2009, a large family of FHE

schemes has emerged, most of them relying on the hardness of lattice problems such as RLWE (Ring Learning With Errors) [42,49]. These schemes are often distinguished by the message spaces they natively support. For instance, BGV, BFV and GBFV [9,10,27,30] target arithmetic over finite field vectors, while FHEW and TFHE [17,26] are tailored to fast evaluation of binary or small-integer circuits. By contrast, CKKS [16] is designed for approximate arithmetic over complex vectors.

**CKKS Scheme: From Approximate to Exact.** The CKKS scheme [16] is best known for high-throughput SIMD-style (single-instruction–multiple-data, [48]) approximate arithmetic on encrypted complex numbers. It can encrypt vectors $\boldsymbol{z} \in \mathbb{C}^n$ and supports slotwise addition and multiplication, as well as conjugation and the application of linear maps. In its classical use, CKKS is an approximate scheme, and an encrypted complex vector is only recovered approximately after decryption. Accordingly, while bootstrapping [31] in other schemes often deals with noise reduction, bootstrapping in the context of CKKS has traditionally been viewed as a way to enable further homomorphic multiplications, without targeting noise reduction. This line of work has led to a rich literature; see *e.g.* [11,14,15,20,21,36].

For a long time, the approximate viewpoint on CKKS suggested that the scheme was poorly suited to exact computation: the approximation error was thought to be inseparable from the message itself. Recent work [25] challenged this intuition by showing that, for discrete message sets, the error can instead be viewed as a small perturbation around well-separated message points and contracted homomorphically. In particular, [25] demonstrated exact bit processing in CKKS using low-degree noise-cleaning polynomials (*e.g.*, $x \mapsto 3x^2 - 2x^3$), which fix 0 and 1 and have vanishing derivative at these points, thereby reducing noise while preserving the encrypted bits. Follow-up works extended this paradigm beyond bits to small integers and roots of unity, and developed dedicated discrete (functional) bootstrapping techniques for these settings, *e.g.*, [2,3,4,18,44]; extensions to larger integer message sets were also studied, *e.g.*, [29,37,38]. Taken together, these advances have positioned CKKS as a practical platform for exact SIMD computation on discrete message sets.

**Why Finite-Field Arithmetic Beyond BGV/BFV?** Many cryptographic computations are natively defined over finite fields, including symmetric primitives such as AES [23,43], message authentication over $\mathbb{F}_{2^{128}}$, and post-quantum signatures such as MAYO [5]. Finite fields also appear in recent proof systems, for instance in SNARKs over towers of binary fields [24]. There is growing interest in evaluating such primitives under FHE, from classical transciphering applications to more recent constructions such as thresholdized signatures [46], one-round blind signatures [6], and blind zk-SNARK protocols [28]. These applications rely on efficient homomorphic arithmetic over fields $\mathbb{F}_{p^r}$.

The natural high-throughput baselines for such computations are BGV and BFV. However, their efficiency depends strongly on the algebraic compatibility

between the field $\mathbb{F}_{p^r}$ and the chosen cyclotomic ring. For some fields, this forces wasteful packing regimes, as is for instance the case for $\mathbb{F}_{2^{128}}$. Moreover, although BGV/BFV provide high throughput, their bootstrapping latency can be substantial, which is problematic when only a few further operations are needed. This motivates the search for a complementary approach: one that retains SIMD-style throughput, is less sensitive to algebraic properties of $(p, r)$, and achieves lower latency.

**Our Contribution.** We address the above challenges by extending the emerging discrete CKKS toolbox to arithmetic over finite fields $\mathbb{F}_{p^r}$ with small characteristic $p$. Our starting point is a new representation of field elements as complex vectors, which we call *spectral encoding*. It is tailored to the native SIMD semantics of CKKS: with spectral encoding, homomorphic addition and multiplication of CKKS ciphertexts directly implement addition and multiplication in $\mathbb{F}_{p^r}$. This gives a natural way to evaluate $\mathbb{F}_{p^r}$-circuits homomorphically in CKKS.

A standard alternative is to identify $\mathbb{F}_{p^r}$ with the slot algebra $\mathbb{F}_p^r$ and represent field elements by $r$ slots in $\mathbb{F}_p$. We call this representation the *coefficient encoding* (since the slots correspond to polynomial coefficients). Our construction keeps this representation available: we provide an efficient homomorphic switch between spectral and coefficient encodings. This allows multiplication-heavy parts to be evaluated in spectral encoding, while $\mathbb{F}_p$-linear maps are handled more naturally in coefficient encoding.

Our implementation in Lattigo [50] shows that the proposed framework is practical and insensitive to the algebraic properties of the field. Compared with BGV-based implementations, it achieves speedups of up to 178× in amortized multiplication time when bootstrapping is taken into account, with the largest gains appearing for small characteristic $p$ and large extension degree $r$. It also achieves substantially lower latency.

**Related Work.** As discussed above, BGV and BFV are the natural high-throughput baselines for exact finite-field arithmetic under FHE. They support computations over discrete plaintext rings and fields, including structures such as $\mathbb{Z}/p^r\mathbb{Z}$ and $\mathbb{F}_{p^r}$; the generalized framework of GBFV [30] primarily targets large-characteristic fields. Recent work has also shown that CKKS can support exact computation on large discrete message sets of the form $\mathbb{Z}/p^r\mathbb{Z}$, primarily implemented for $p = 2$ [29,37,38]. However, arithmetic over the ring $\mathbb{Z}/p^r\mathbb{Z}$ is structurally different from arithmetic over the extension field $\mathbb{F}_{p^r}$. To the best of our knowledge, this is the first work proposing arithmetic over extension fields $\mathbb{F}_{p^r}$ within CKKS.

### 1.2 Detailed Contribution Overview

We introduce a natural method for realizing finite-field arithmetic homomorphically through SIMD-style operations over complex slots in CKKS. Our approach uses the ability to homomorphically evaluate in complex quotient rings $\mathbb{C}[X]/(F)$

for polynomials $F \in \mathbb{C}[X]$ within CKKS. Similar techniques have recently appeared [21,35,38], typically for polynomials $F$ of the form $X^n + 1$ or divisors thereof. However, these works aim to prevent wrap-around modulo $F$ by keeping polynomial degrees small. In contrast, we deliberately exploit reduction modulo $F$ as is done in [29], and turn it into the mechanism that realizes finite-field arithmetic.

For concreteness, we illustrate our approach on the 16-element field $\mathbb{F}_{16}$; the general case of $\mathbb{F}_{p^r}$ for small primes $p$ and small exponents $r$ will be treated later. Recall that $\mathbb{F}_{16} = \mathbb{F}_{2^4}$ can be represented as a quotient $\mathbb{F}_{16} = \mathbb{F}_2[X]/(f)$, where $f \in \mathbb{F}_2[X]$ is an irreducible polynomial of degree four, say $f = X^4 + X + 1$. Every element $a \in \mathbb{F}_{16}$ then has a unique representation $a = a_0 + a_1X + a_2X^2 + a_3X^3$ with coefficients $a_i \in \mathbb{F}_2$. To homomorphically compute with such elements within CKKS, we lift $f$ to the integer polynomial $F = X^4 + X + 1 \in \mathbb{Z}[X]$ and regard $a$ as an element $\hat{a} \in \mathbb{Z}[X]/(F)$ whose reduction modulo 2 equals $a$ (for instance, by interpreting the coefficients $a_i$ as integers 0 or 1). Since $\mathbb{Z}[X]/(F)$ embeds into $\mathbb{C}[X]/(F)$, this perspective allows us to manipulate elements $a$ directly within CKKS, as we will now explain.

**Spectral Encoding.** We begin by explaining how elements of $\mathbb{F}_{16}$ can be represented as complex vectors, a format we refer to as *spectral encoding*. Given an element $a \in \mathbb{F}_{16}$, we first choose a lift $\hat{a} \in \mathbb{Z}[X]/(F)$ whose reduction modulo 2 equals $a$. This lift may then be viewed naturally inside the quotient ring $\mathbb{C}[X]/(F)$. Prior work typically works with the coefficient vector $[\hat{a}]$ of $\hat{a}$, which we call the *coefficient encoding* of $a$ and denote by $\eta(a) = [\hat{a}] \in \mathbb{Z}^4$. This format is convenient but not SIMD-friendly for multiplication: in general, for $a, b \in \mathbb{F}_{16}$ we have $\eta(ab) \neq \eta(a) \odot \eta(b)$, where $\odot$ denotes componentwise multiplication. Instead, we propose to use a different encoding technique. Onto $\hat{a}$, we apply the isomorphism $\varphi : \mathbb{C}[X]/(F) \xrightarrow{\sim} \mathbb{C}^4$ defined by evaluation at the four complex roots of $F$, namely $\omega_0 \approx -0.73 - 0.43\mathrm{i}$, $\omega_1 \approx -0.73 + 0.43\mathrm{i}$, $\omega_2 \approx 0.73 - 0.93\mathrm{i}$ and $\omega_3 \approx 0.73 + 0.93\mathrm{i}$, where $\mathrm{i} \in \mathbb{C}$ (upright font) denotes the imaginary unit. This yields a complex vector $\varphi(\hat{a}) = (\hat{a}(\omega_i))_{i \in \{0,1,2,3\}} \in \mathbb{C}^4$ as illustrated in Figure 1, which we denote by $\varepsilon(a)$ and call the *spectral encoding* of $a$. Spectral encoding thus defines a rule $\varepsilon : \mathbb{F}_{16} \to \mathbb{C}^4$ (though, since it depends on the choice of lift $\hat{a}$, it is multivalued). Note that we distinguish between $\varepsilon$ and $\varphi$ because $\varepsilon$ is applied to field elements, while $\varphi$ is applied to complex polynomials modulo $F$.

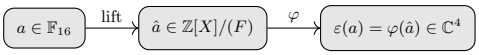


**Fig. 1.** Pipeline for spectral encoding.

As an example, for $a = 1 + X^3 \in \mathbb{F}_{16}$, we can consider the lift $\hat{a} = 3 - 5X^3 \in \mathbb{Z}[X]/(F)$ (though in practice, it is preferable to choose small-coefficient lifts for

numerical stability). In this case, the encoded vector is given by:

$$\varepsilon\big(1+X^3\big) = \begin{pmatrix} 3-5\omega_0^3 \\ 3-5\omega_1^3 \\ 3-5\omega_2^3 \\ 3-5\omega_3^3 \end{pmatrix} \in \mathbb{C}^4.$$

Observe that $\varepsilon\big(1+X^3\big)$ can also be computed in terms of a matrix–vector multiplication, where the matrix is the Vandermonde matrix generated by the roots $\omega_i$:

$$\varepsilon\big(1+X^3\big) = \begin{pmatrix} 1 & \omega_0 & \omega_0^2 & \omega_0^3 \\ 1 & \omega_1 & \omega_1^2 & \omega_1^3 \\ 1 & \omega_2 & \omega_2^2 & \omega_2^3 \\ 1 & \omega_3 & \omega_3^2 & \omega_3^3 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 0 \\ 0 \\ -5 \end{pmatrix}.$$

Note that $(3,0,0,-5) = \eta(a)$ is a coefficient encoding of $a$. Therefore, if we denote the above $4\times 4$ Vandermonde matrix by $\boldsymbol{M}_{\varepsilon\leftarrow\eta} \in \mathbb{C}^{4\times 4}$, then it holds that $\varepsilon(a) = \boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot \eta(a)$ for every $a \in \mathbb{F}_{16}$. Equivalently, if we set $\boldsymbol{M}_{\eta\leftarrow\varepsilon} = \boldsymbol{M}_{\varepsilon\leftarrow\eta}^{-1}$, then we have $\eta(a) = \boldsymbol{M}_{\eta\leftarrow\varepsilon} \cdot \varepsilon(a)$. We call these two matrices the *encoding switch matrices*. To retrieve $a$ from its spectral encoding $\boldsymbol{z} = \varepsilon(a) \in \mathbb{C}^4$, one can first apply the inverse isomorphism $\varphi^{-1}$, then reduce the resulting element modulo two: $\varphi^{-1}(\boldsymbol{z}) \bmod 2 = a$.

**SIMD-Friendly.** The crucial advantage of the spectral encoding $\varepsilon$ over the coefficient encoding $\eta$ is that it is SIMD-friendly. Indeed, for any $a, b \in \mathbb{F}_{16}$, we have:

$$\varepsilon(a) \odot \varepsilon(b) = \varphi(\hat{a}) \odot \varphi(\hat{b}) = \varphi(\hat{a} \cdot \hat{b}) = \varphi\Big(\widehat{ab}\Big) = \varepsilon(ab),$$

and a similar computation shows that addition is also respected. As a result, since CKKS allows componentwise addition and multiplication of encrypted complex vectors, we can use CKKS to perform addition and multiplication in $\mathbb{F}_{16}$ homomorphically. Since CKKS ciphertexts can hold up to $N/2$ complex slots, we can pack $N/8$ elements of $\mathbb{F}_{16}$ into a single ciphertext.

**Frobenius Powers.** Assume that we are given a spectral encoding $\varepsilon(a)$ for some $a \in \mathbb{F}_{16}$ and that we wish to compute $\varepsilon\big(a^8\big)$. One option is to use successive squarings, which requires multiplicative depth 3. However, the 8-th power can in fact be computed in a single multiplicative level, and in certain situations at essentially no additional cost. To understand this, consider an element $a = a_0 + a_1X + a_2X^2 + a_3X^3 \in \mathbb{F}_{16}$ with coefficient vector $[a] = (a_0, a_1, a_2, a_3) \in \mathbb{F}_2^4$. The map $\mathsf{Pow}_3 : a \mapsto a^{2^3}$ is $\mathbb{F}_2$-linear, which means that we can get $\big[a^8\big]$ from $[a]$ by left-multiplying it by a matrix over $\mathbb{F}_2$. Explicitly, we have:

$$\big[a^8\big] = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot [a].$$

Viewing this matrix over $\mathbb{Z}$ and denoting it by $\boldsymbol{M}_{\eta\leftarrow\eta}(\mathsf{Pow}_3)$, we therefore have $\eta\big(a^8\big) = \boldsymbol{M}_{\eta\leftarrow\eta}(\mathsf{Pow}_3) \cdot \eta(a)$. Using the encoding switch matrices, we also get:

$$\varepsilon\big(a^8\big) = \boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot \boldsymbol{M}_{\eta\leftarrow\eta}(\mathsf{Pow}_3) \cdot \boldsymbol{M}_{\eta\leftarrow\varepsilon} \cdot \varepsilon(a).$$

Therefore, 8-th powers can be computed directly in spectral encoding by left multiplication with the following complex matrix, at the cost of a single multiplicative level:

$$\boldsymbol{M}_{\varepsilon\leftarrow\varepsilon}(\mathsf{Pow}_3) = \boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot \boldsymbol{M}_{\eta\leftarrow\eta}(\mathsf{Pow}_3) \cdot \boldsymbol{M}_{\eta\leftarrow\varepsilon} \in \mathbb{C}^{4\times 4}.$$

The same argument applies to the maps $a \mapsto a^2$ and $a \mapsto a^4$. Furthermore, because these power maps are realized as matrix multiplications, they can be fused with other matrix operations (*e.g.*, during bootstrapping), making the computation of these powers often free in practice.

**Bootstrapping.** In our setting, bootstrapping serves three purposes simultaneously: it refreshes the ciphertext modulus, reduces the accumulated CKKS noise, and re-centers the lifted coefficients modulo 2 (or more generally modulo $p$). The last point is important because large lifted coefficients directly amplify the noise during subsequent homomorphic multiplications.

Concretely, suppose we start from a ciphertext encrypting $\eta(a) + \boldsymbol{e}$ in the slots for some $a \in \mathbb{F}_{16}$, where $\boldsymbol{e} \in \mathbb{C}^4$ is an error term. Applying a bit-oriented bootstrapping procedure such as XBOOT [44] refreshes the modulus, reduces the noise magnitude, and reduces the lifted coefficients modulo 2. The resulting ciphertext therefore encrypts $\eta(a) \bmod 2 + \boldsymbol{e}'$, where $\|\boldsymbol{e}'\|_\infty \ll \|\boldsymbol{e}\|_\infty$.

Starting instead from a spectral encryption $\varepsilon(a)$, we first homomorphically apply the linear map $\boldsymbol{M}_{\eta\leftarrow\varepsilon}$ to recover $\eta(a)$ in the slots before bootstrapping. Since bootstrapping procedures typically begin with the linear slot-to-coefficient step, this transformation can be merged into the bootstrapping itself at no additional cost, thereby saving one multiplicative level. The same approach extends to other small characteristics: for $p > 2$, one may for instance use the small-integer bootstrapping SI-BTS [4].

**Experimental Results.** We implemented our framework in Lattigo [50] and compare its bootstrapping cost with the analogous BGV pipeline in HElib [33].[1] Since bootstrapping dominates the running time, we report amortized costs per packed finite-field element and per usable multiplicative level recovered by bootstrapping. All benchmarks were run single-threaded on a MacBook Pro with an M4 chip (14 CPU cores and 24 GB of RAM). Full details appear in Section 9. As summarized in Table 1, our approach achieves a lower amortized cost, yielding clear speedups over BGV.

[1] The code will be made publicly available upon publication.

**Table 1.** Amortized bootstrapping time in milliseconds per packed element of $\mathbb{F}_{p^r}$ and per usable multiplicative level.

| | Amortized time in ms (time/elem./levels) | | |
|---|---|---|---|
| $p^r$ | BGV | Ours | Speedup |
| $2^8$ | 11.9 | 0.317 | 38× |
| $2^{128}$ | 2 470 | 13.9 | 178× |
| $31^3$ | 1.07 | 0.636 | 1.7× |

## 2 Notation

**Sets.** We write $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ for the integers, rationals, real numbers, and complex numbers, respectively. Given a prime $p$ and an extension degree $r \geqslant 1$, we write $\mathbb{F}_p$ for the field with $p$ elements and $\mathbb{F}_{p^r}$ for its degree-$r$ extension. For $n \geqslant 1$, we write $\mathbb{Z}/n\mathbb{Z}$ for the ring of integers modulo $n$. For odd $n \geqslant 1$, we use the centered representatives $\mathbb{Z}_n = \{-(n-1)/2, \ldots, (n-1)/2\} \subseteq \mathbb{Z}$, while for even $n \geqslant 2$, we use the representatives $\mathbb{Z}_n = \{-n/2+1, \ldots, n/2\} \subseteq \mathbb{Z}$. Given integers $a \leqslant b$, we write $[\![a, b]\!] = \{a, a+1, \ldots, b\}$, and for $n \geqslant 1$ we abbreviate $[\![n]\!] = \{0, 1, \ldots, n-1\}$. For a set of integers $S \subseteq \mathbb{Z}$, we write $S \bmod n = \{s \bmod n : s \in S\} \subseteq [\![n]\!]$. An interval modulo $n$ means a set of the form $[\![a, b]\!] \bmod n$ for some integers $a \leqslant b$; its length is its cardinality. Given sets $S, T \subseteq \mathbb{C}$, we write $S + T = \{s + t : s \in S, t \in T\}$.

**Arithmetic.** We denote the imaginary unit by $\mathrm{i} \in \mathbb{C}$ (upright font) and for an integer $n \geqslant 1$, we write $\zeta_n = \exp(2\pi\mathrm{i}/n)$ for the primitive $n$-th root of unity with argument $2\pi/n$. Euler's totient function is denoted by $\phi$. For integers $n_0, \ldots, n_{k-1} \geqslant 1$, we write $\mathrm{lcm}(n_0, \ldots, n_{k-1})$ for their least common multiple. If $a$ and $b$ are coprime integers, we write $\mathrm{ord}_b(a)$ for the multiplicative order of $a$ modulo $b$, that is, the smallest positive integer $k$ such that $a^k \equiv 1 \bmod b$. Given a ring $R$, we denote by $R^\times$ the group of units of $R$.

**Vectors and Matrices.** For a ring $R$, we write $R^n$ for length-$n$ vectors over $R$ and $R^{m \times n}$ for $m \times n$ matrices over $R$. Vectors are always regarded as column vectors, even when written in tuple notation for compactness. Vectors and matrices are denoted in bold. For square matrices $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_{k-1}$ over $R$, we write $\mathsf{Diag}(\boldsymbol{M}_0, \ldots, \boldsymbol{M}_{k-1})$ for the corresponding block-diagonal matrix. We see $R^n$ as a ring under componentwise addition ($+$) and multiplication ($\odot$). For $\boldsymbol{z} = (z_i)_{i \in [\![n]\!]} \in \mathbb{C}^n$, we denote its infinity norm by $\|\boldsymbol{z}\|_\infty = \max_{i \in [\![n]\!]} |z_i|$. We define the (cyclic) rotation of $\boldsymbol{z}$ by $j \in \mathbb{Z}$ slots as $\mathsf{Rot}_j(\boldsymbol{z}) = (z_{(i+j) \bmod n})_{i \in [\![n]\!]} \in \mathbb{C}^n$, and the complex conjugate of $\boldsymbol{z}$ as $\overline{\boldsymbol{z}} = (\overline{z_i})_{i \in [\![n]\!]} \in \mathbb{C}^n$. As is standard in CKKS, if $m$ is a multiple of $n$, we embed $\mathbb{C}^n$ into $\mathbb{C}^m$ by repeating slots, $\boldsymbol{z} \mapsto (\boldsymbol{z}, \ldots, \boldsymbol{z})$ (with $m/n$ copies), so expressions like $\boldsymbol{z} + \boldsymbol{w}$ with $\boldsymbol{z} \in \mathbb{C}^n$ and $\boldsymbol{w} \in \mathbb{C}^m$ are unambiguous.

**Polynomials.** For a ring $R$, we write $R[X]$ for polynomials in $X$ with coefficients in $R$; polynomials are not bolded. For $F \in R[X]$, we denote by $R[X]/(F)$ the quotient of $R[X]$ by the ideal generated by $F$, and for $P, Q \in R[X]/(F)$ we write $P \cdot Q$ for their product (with reduction modulo $F$ implicit). Given a polynomial $P = \sum_{i=0}^{n-1} P_i X^i \in R[X]$, we define its coefficient vector as $[P] = (P_i)_{i \in [\![n]\!]} \in R^n$. If $P \in R[X]/(F)$ with $\deg(F) = n$, then $[P] \in R^n$ denotes the coefficient vector of the unique representative of $P$ of degree $< n$. When $P \in \mathbb{C}[X]$ or $P \in \mathbb{C}[X]/(F)$, we define $\|P\|_\infty = \|[P]\|_\infty$. When $P \in \mathbb{F}_p[X]$, we define $\|P\|_\infty = \|Q\|_\infty$, where $Q$ is the unique polynomial in $\mathbb{Z}_p[X]$ which satisfies $Q \bmod p = P$.

**Miscellaneous.** The symbols $\mathbb{P}$, $\mathbb{E}$, and Var are used to denote probability, expectation, and variance, respectively. We write $\lfloor \cdot \rceil$ for (componentwise and coefficientwise) rounding to the nearest integer, with ties broken arbitrarily.

## 3 Background on CKKS

We recall the components of the CKKS scheme [16] that are relevant for our construction. The security of the scheme is based on the hardness of the Ring Learning With Errors (RLWE) problem [42,49].

**Encoding and Decoding.** We fix a power of two $N$ and define the rings $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{S} = \mathbb{R}[X]/(X^N + 1)$. CKKS supports approximate arithmetic on complex vectors. Such vectors are first mapped to a polynomial in $\mathcal{R}$ before encryption. This mapping is defined via the canonical embedding $\mathsf{DFT} : \mathcal{S} \xrightarrow{\sim} \mathbb{C}^{N/2}$, given by:

$$\mathsf{DFT}(P) = \left(P\left(\zeta_{2N}^{5^i}\right)\right)_{i \in [\![N/2]\!]} \in \mathbb{C}^{N/2},$$

where $\zeta_{2N} = \exp(\pi \mathrm{i}/N)$ is a primitive $(2N)$-th root of unity. To encode a vector $\boldsymbol{z} \in \mathbb{C}^{N/2}$, one applies $\mathsf{DFT}^{-1}$ to obtain a polynomial in $\mathcal{S}$, scales by a fixed factor $\Delta \gg 0$, and rounds coefficientwise:

$$\mathsf{Ecd}(\boldsymbol{z}) = \lfloor \Delta \cdot \mathsf{DFT}^{-1}(\boldsymbol{z}) \rceil \in \mathcal{R}.$$

Decoding a polynomial $m \in \mathcal{R}$ is performed as $\mathsf{Dcd}(m) = \mathsf{DFT}(m)/\Delta \in \mathbb{C}^{N/2}$.

**Ciphertexts.** A CKKS ciphertext at modulus $q$ is a pair $\mathbf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1) \in \mathcal{R}_q^2$, where $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. It encrypts a polynomial $m \in \mathcal{R}$ under a secret key $s \in \mathcal{R}$ with implicit error $e \in \mathcal{R}$ if $\|m+e\|_\infty < q/2$ and if it satisfies $\mathsf{ct}_0 + \mathsf{ct}_1 \cdot s \equiv m+e \bmod q$. In that case, we write $\mathbf{ct} = \mathsf{Enc}_s(m+e)$, or simply $\mathbf{ct} \approx \mathsf{Enc}_s(m)$. When the secret key is clear from the context, we omit the subscript. We also use the combined notation $\mathbf{ct} \approx \mathsf{Enc}(m+e)$ to mean that the hidden encryption error is negligible compared to the explicit error term $e$. Finally, we write $\mathsf{EncEcd}(\boldsymbol{z})$ as shorthand for $\mathsf{Enc}(\mathsf{Ecd}(\boldsymbol{z}))$. If $\mathbf{ct} \approx \mathsf{EncEcd}(\boldsymbol{z})$, we say that $\mathbf{ct}$ encrypts $\boldsymbol{z}$.

**Elementary Operations.** We consider ciphertexts $\mathbf{ct}, \mathbf{ct}_0, \mathbf{ct}_1 \in \mathcal{R}_q^2$ under the same secret key $s \in \mathcal{R}$, satisfying $\mathbf{ct} \approx \mathsf{EncEcd}(\boldsymbol{z})$, $\mathbf{ct}_0 \approx \mathsf{EncEcd}(\boldsymbol{z}_0)$, and $\mathbf{ct}_1 \approx \mathsf{EncEcd}(\boldsymbol{z}_1)$ for some vectors $\boldsymbol{z}, \boldsymbol{z}_0, \boldsymbol{z}_1 \in \mathbb{C}^{N/2}$. Below, $q'$ denotes a modulus with $q' \approx q/\Delta$. We recall the basic homomorphic operations supported by CKKS:

- *Key switching.* One can switch from secret key $s$ to another $s' \in \mathcal{R}$ by computing $\mathbf{ct}_{\mathsf{switch}} = \mathsf{KeySwitch}_{s \mapsto s'}(\mathbf{ct}) \in \mathcal{R}_q^2$; it requires a key-switching key $\mathsf{swk}_{s \mapsto s'}$ generated during key generation. The output satisfies $\mathbf{ct}_{\mathsf{switch}} \approx \mathsf{EncEcd}_{s'}(\boldsymbol{z})$.
- *Addition and subtraction.* We can add and subtract ciphertexts componentwise: $\mathbf{ct}_0 \pm \mathbf{ct}_1 \in \mathcal{R}_q^2$ satisfy $\mathbf{ct}_0 \pm \mathbf{ct}_1 \approx \mathsf{EncEcd}(\boldsymbol{z}_0 \pm \boldsymbol{z}_1)$.
- *Ciphertext–ciphertext multiplication.* We can compute a product followed by rescaling: $\mathbf{ct}_{\mathsf{prod}} = \mathbf{ct}_0 \times \mathbf{ct}_1 \in \mathcal{R}_{q'}^2$ satisfies $\mathbf{ct}_{\mathsf{prod}} \approx \mathsf{EncEcd}(\boldsymbol{z}_0 \odot \boldsymbol{z}_1)$. This operation requires a key-switching key $\mathsf{swk}_{s^2 \mapsto s}$. (We use $\times$ to emphasize that rescaling is performed, hence the modulus drop from $q$ to $q'$.)
- *Plaintext–ciphertext multiplication.* For a known vector $\boldsymbol{w} \in \mathbb{C}^{N/2}$, we can multiply homomorphically by $\boldsymbol{w}$: $\mathbf{ct}_{\mathsf{prod}} = \mathsf{Ecd}(\boldsymbol{w}) \times \mathbf{ct} \in \mathcal{R}_{q'}^2$ yields $\mathbf{ct}_{\mathsf{prod}} \approx \mathsf{EncEcd}(\boldsymbol{w} \odot \boldsymbol{z})$. If $\mathsf{DFT}^{-1}(\boldsymbol{w}) \in \mathcal{R}$, then we can instead compute $\mathbf{ct}_{\mathsf{prod}} = \mathsf{DFT}^{-1}(\boldsymbol{w}) \cdot \mathbf{ct} \in \mathcal{R}_q^2$ (without reducing the modulus) and still obtain $\mathbf{ct}_{\mathsf{prod}} \approx \mathsf{EncEcd}(\boldsymbol{w} \odot \boldsymbol{z})$.
- *Conjugation.* We can homomorphically conjugate slots: $\mathbf{ct}_{\mathsf{conj}} = \mathsf{Conj}(\mathbf{ct}) \in \mathcal{R}_q^2$ satisfies $\mathbf{ct}_{\mathsf{conj}} \approx \mathsf{EncEcd}(\overline{\boldsymbol{z}})$. This involves key switching from $s' = s\big(-X^{N-1}\big)$ back to $s$ and therefore requires $\mathsf{swk}_{s' \mapsto s}$.
- *Rotation.* We can homomorphically rotate slots by $i \in [\![N/2]\!]$ positions: $\mathbf{ct}_{\mathsf{rot}} = \mathsf{Rot}_i(\mathbf{ct}) \in \mathcal{R}_q^2$ satisfies $\mathbf{ct}_{\mathsf{rot}} \approx \mathsf{EncEcd}(\mathsf{Rot}_i(\boldsymbol{z}))$. This operation also uses key switching, from $s' = s\big(X^j\big)$ to $s$ with $j = 5^i$, hence requires $\mathsf{swk}_{s' \mapsto s}$.

**Matrix Multiplication.** We recall how to left-multiply an encrypted vector $\mathbf{ct} \approx \mathsf{EncEcd}(\boldsymbol{z})$ with $\boldsymbol{z} \in \mathbb{C}^n$ by a plaintext matrix $\boldsymbol{M} = (M_{i,j})_{i,j \in [\![n]\!]} \in \mathbb{C}^{n \times n}$ using the *diagonal method* [32]. For $j \in [\![n]\!]$, define the $j$-th diagonal of $\boldsymbol{M}$ as $\mathsf{Diag}_j(\boldsymbol{M}) = (M_{i,(i+j) \bmod n})_{i \in [\![n]\!]} \in \mathbb{C}^n$ and its diagonal support as $\mathsf{DiagSupp}(\boldsymbol{M}) = \{j \in [\![n]\!] : \mathsf{Diag}_j(\boldsymbol{M}) \neq \mathbf{0}\}$. We encode $\boldsymbol{M}$ as $\mathsf{Ecd}(\boldsymbol{M}) = (\mathsf{Ecd}(\mathsf{Diag}_j(\boldsymbol{M})))_{j \in \mathsf{DiagSupp}(\boldsymbol{M})}$. This allows us to compute:

$$\mathsf{Ecd}(\boldsymbol{M}) \times \mathbf{ct} = \sum_{j \in \mathsf{DiagSupp}(\boldsymbol{M})} \mathsf{Ecd}(\mathsf{Diag}_j(\boldsymbol{M})) \times \mathsf{Rot}_j(\mathbf{ct}),$$

which satisfies $\mathsf{Ecd}(\boldsymbol{M}) \times \mathbf{ct} \approx \mathsf{EncEcd}(\boldsymbol{M}\boldsymbol{z})$. This requires $|\mathsf{DiagSupp}(\boldsymbol{M})|$ plaintext–ciphertext multiplications and the same number of rotations (one rotation can be saved if $0 \in \mathsf{DiagSupp}(\boldsymbol{M})$). Further reductions are possible using *baby-step giant-step techniques*, *e.g.*, [14]. For instance, if $\mathsf{DiagSupp}(\boldsymbol{M})$ is contained in a set of $D$ integers forming an arithmetic progression modulo $n$, then we can compute $\mathsf{Ecd}(\boldsymbol{M}) \times \mathbf{ct}$ with $\mathcal{O}(D)$ plaintext–ciphertext multiplications and $\mathcal{O}\big(\sqrt{D}\big)$ rotations.

# 4 Finite-Field Arithmetic from Complex Vectors

In this section, we describe our general method of encoding finite-field elements within complex vectors, and this will later on allow finite-field arithmetic in CKKS. From now on, $p$ will always denote a small prime number and $r \geqslant 1$ a small extension degree (for instance, we performed successful tests for $p$ and $r$ up to 6 and 7 bits, respectively). We want to perform arithmetic in the finite field $\mathbb{F}_{p^r} = \mathbb{F}_p[X]/(f)$, where $f \in \mathbb{F}_p[X]$ is any monic irreducible polynomial of degree $r$.

## 4.1 Spectral Encoding

To encode an element $a \in \mathbb{F}_{p^r}$ as a complex vector $\varepsilon(a) \in \mathbb{C}^r$, consider the lift $F \in \mathbb{Z}[X]$ of $f$ with coefficients in $\mathbb{Z}_p$ and roots $\omega_i \in \mathbb{C}$ for $i \in [\![r]\!]$. These roots are distinct:

**Proposition 1.** *Let $f \in \mathbb{F}_p[X]$ be an irreducible polynomial. Then, any lift $F \in \mathbb{Z}[X]$ of $f$ with the same degree as $f$ has distinct complex roots.*

A proof is provided in Appendix A. It follows that evaluation at the roots $\omega_i$ of $F$ defines a $\mathbb{C}$-algebra isomorphism $\varphi : \mathbb{C}[X]/(F) \to \mathbb{C}^r$, explicitly given by $\varphi(P) = (P(\omega_i))_{i \in [\![r]\!]}$ and typically referred to as the *spectral isomorphism* of $\mathbb{C}[X]/(F)$. In particular, $\varphi$ is a $\mathbb{C}$-linear isomorphism and can be represented as an invertible matrix over $\mathbb{C}$ (with respect to the monomial basis of $\mathbb{C}[X]/(F)$ and the standard basis of $\mathbb{C}^r$). We will return to this matrix representation in Section 4.2.

From now on, we fix the polynomials $f$ and $F$ as well as the roots $\omega_i$ and the isomorphism $\varphi$. For an element $a \in \mathbb{F}_{p^r}$, we denote by $\hat{a} \in \mathbb{Z}[X]/(F)$ any lift of $a$, that is, any element which satisfies $\hat{a} \bmod p = a$. The *spectral encoding* of $a \in \mathbb{F}_{p^r}$ is a complex vector of the following form:

$$\varepsilon(a) = \varphi(\hat{a}) = (\hat{a}(\omega_i))_{i \in [\![r]\!]} \in \mathbb{C}^r.$$

We refer to Figure 1 for an illustration of the encoding pipeline, where $\mathbb{F}_{16}$ and $\mathbb{C}^4$ are replaced by $\mathbb{F}_{p^r}$ and $\mathbb{C}^r$, respectively. Decoding $\boldsymbol{z} = \varepsilon(a) \in \mathbb{C}^r$ back to $a \in \mathbb{F}_{p^r}$ is done by computing $\varphi^{-1}(\boldsymbol{z}) \bmod p = a$. The spectral encoding $\varepsilon$ is therefore based on the diagram of ring morphisms presented in Figure 2. Note that $\varepsilon(a)$ depends on the choice of the lift $\hat{a}$, and $\varepsilon$ is thus a multivalued map.

$$\mathbb{F}_{p^r} = \frac{\mathbb{F}_p[X]}{(f)} \twoheadleftarrow \frac{\mathbb{Z}[X]}{(F)} \hookrightarrow \frac{\mathbb{C}[X]}{(F)} \xrightarrow{\sim} \mathbb{C}^r$$

**Fig. 2.** Diagram of ring morphisms underlying the spectral encoding $\varepsilon$, where $\twoheadrightarrow$, $\hookrightarrow$ and $\xrightarrow{\sim}$ denote surjective, injective and bijective morphisms, respectively.

**Proposition 2.** *Let $a, b \in \mathbb{F}_{p^r}$. Then, the following properties hold:*

$$\varepsilon(a) + \varepsilon(b) = \varepsilon(a+b), \qquad \varepsilon(a) \odot \varepsilon(b) = \varepsilon(ab).$$

*Proof.* We have $\varepsilon(a)+\varepsilon(b) = \varphi(\hat{a})+\varphi(\hat{b}) = \varphi(\hat{a}+\hat{b}) = \varphi\Big(\widehat{a+b}\Big) = \varepsilon(a+b)$, and a similar computation shows the multiplication property. □

### 4.2 Spectral Encoding versus Coefficient Encoding

So far, for an element $a \in \mathbb{F}_{p^r}$, we have worked with a spectral encoding $\varepsilon(a)$, obtained by evaluating a lift of the polynomial $a$ at the complex roots $\omega_i$ of the lift $F$ of $f$. We also consider an alternative representation of $a$, which we call the *coefficient encoding* and which we define simply as $\eta(a) = [\hat{a}] \in \mathbb{Z}^r$. Its advantage is that the $r$ coefficients of $a$ appear explicitly in the slots, though this format is only SIMD-friendly with respect to addition and not with respect to multiplication.

Let us now explain how to move from a spectral encoding $\varepsilon(a)$ to a coefficient encoding $\eta(a)$, and vice versa. To do so, we consider the Vandermonde matrix associated with the roots $\omega_i$ of $F$, which we denote by $\boldsymbol{M}_{\varepsilon\leftarrow\eta} \in \mathbb{C}^{r\times r}$. Explicitly, the entry in row $i \in [\![r]\!]$ and column $j \in [\![r]\!]$ of $\boldsymbol{M}_{\varepsilon\leftarrow\eta}$ is set to $\omega_i^j$. We further denote by $\boldsymbol{M}_{\eta\leftarrow\varepsilon} = \boldsymbol{M}_{\varepsilon\leftarrow\eta}^{-1}$ its inverse. The following result, essentially following from the commutative diagrams in Figure 3, shows that these two matrices allow us to switch between the spectral and coefficient encodings, which is why we refer to them as the *encoding-switch matrices*:

**Lemma 1.** *For every $a \in \mathbb{F}_{p^r}$, it holds that:*

$$\boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot \eta(a) = \varepsilon(a), \qquad \boldsymbol{M}_{\eta\leftarrow\varepsilon} \cdot \varepsilon(a) = \eta(a).$$

*Proof.* Let $P = \sum_{j\in[\![r]\!]} P_j X^j \in \mathbb{C}[X]/(F)$, so that $[P] = (P_j)_{j\in[\![r]\!]} \in \mathbb{C}^r$. For every $i \in [\![r]\!]$, the $i$-th entry of $\boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot [P]$ is $\sum_{j\in[\![r]\!]} \omega_i^j P_j = P(\omega_i)$, and hence $\boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot [P] = (P(\omega_i))_{i\in[\![r]\!]} = \varphi(P)$. Therefore, $\varphi$ is represented by $\boldsymbol{M}_{\varepsilon\leftarrow\eta}$ with respect to the monomial basis of $\mathbb{C}[X]/(F)$ and the standard basis of $\mathbb{C}^r$. Since $\varphi$ is an isomorphism, the matrix $\boldsymbol{M}_{\varepsilon\leftarrow\eta}$ is invertible, and the inverse map $\varphi^{-1}$ is represented by $\boldsymbol{M}_{\eta\leftarrow\varepsilon} = \boldsymbol{M}_{\varepsilon\leftarrow\eta}^{-1}$. This means that $\big[\varphi^{-1}(\boldsymbol{z})\big] = \boldsymbol{M}_{\eta\leftarrow\varepsilon} \cdot \boldsymbol{z}$ for every $\boldsymbol{z} \in \mathbb{C}^r$. Applying these identities to $P = \hat{a}$ gives $\boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot \eta(a) = \boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot [\hat{a}] = \varphi(\hat{a}) = \varepsilon(a)$, and the second relation follows by applying $\boldsymbol{M}_{\eta\leftarrow\varepsilon}$ to both sides. □

*Note 1.* The inverse Vandermonde matrix $\boldsymbol{M}_{\eta\leftarrow\varepsilon}$ can be described through Lagrange interpolation: its $i$-th column is the coefficient vector of the Lagrange polynomial $L_i = \prod_j (X-\omega_j)/(\omega_i-\omega_j) \in \mathbb{C}[X]/(F)$, where the product is taken over $j \in [\![r]\!] \setminus \{i\}$. This description for the Vandermonde inverse is preferable computationally, as it avoids explicitly forming an inverse matrix, which can be numerically unstable.

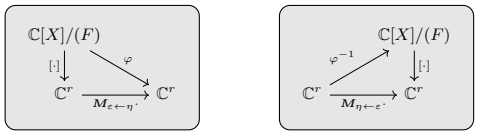


**Fig. 3.** Commutative diagrams for evaluating the spectral isomorphism $\varphi$ and its inverse $\varphi^{-1}$ with $\boldsymbol{M}_{\varepsilon\leftarrow\eta}$ and $\boldsymbol{M}_{\eta\leftarrow\varepsilon}$, respectively.

### 4.3 Applying $\mathbb{F}_p$-Linear Maps

We now describe how to apply an $\mathbb{F}_p$-linear map to an element $a \in \mathbb{F}_{p^r}$ while working with both spectral encodings $\varepsilon(a)$ and coefficient encodings $\eta(a)$. Let $\lambda : \mathbb{F}_{p^r} \to \mathbb{F}_{p^r}$ be an $\mathbb{F}_p$-linear map. We consider the basis $1, X, \ldots, X^{r-1}$ of $\mathbb{F}_{p^r} = \mathbb{F}_p[X]/(f)$, with respect to which we can represent $\lambda$ as a matrix over $\mathbb{F}_p$ of size $r$. Let $\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda) \in \mathbb{Z}^{r\times r}$ denote its integer lift with entries in $\mathbb{Z}_p$, so that $\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)\cdot\eta(a) = \eta(\lambda(a))$ for every $a \in \mathbb{F}_{p^r}$. Further, consider the following three matrices in $\mathbb{C}^{r\times r}$:

$$\begin{aligned}
\boldsymbol{M}_{\varepsilon\leftarrow\eta}(\lambda) &= \boldsymbol{M}_{\varepsilon\leftarrow\eta}\cdot\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda),\\
\boldsymbol{M}_{\eta\leftarrow\varepsilon}(\lambda) &= \boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)\cdot\boldsymbol{M}_{\eta\leftarrow\varepsilon},\\
\boldsymbol{M}_{\varepsilon\leftarrow\varepsilon}(\lambda) &= \boldsymbol{M}_{\varepsilon\leftarrow\eta}\cdot\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)\cdot\boldsymbol{M}_{\eta\leftarrow\varepsilon}.
\end{aligned}$$

**Proposition 3.** *Let $\lambda : \mathbb{F}_{p^r} \to \mathbb{F}_{p^r}$ be an $\mathbb{F}_p$-linear map. Then, for every $a \in \mathbb{F}_{p^r}$, we have:*

$$\begin{aligned}
\boldsymbol{M}_{\varepsilon\leftarrow\eta}(\lambda)\cdot\eta(a) &= \varepsilon(\lambda(a)), & \boldsymbol{M}_{\eta\leftarrow\varepsilon}(\lambda)\cdot\varepsilon(a) &= \eta(\lambda(a)),\\
\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)\cdot\eta(a) &= \eta(\lambda(a)), & \boldsymbol{M}_{\varepsilon\leftarrow\varepsilon}(\lambda)\cdot\varepsilon(a) &= \varepsilon(\lambda(a)).
\end{aligned}$$

*Proof.* The bottom-left relation holds by definition of $\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)$. It remains to prove the other three relations; we show the bottom-right one, the other two being analogous. We need to verify that applying $\boldsymbol{M}_{\varepsilon\leftarrow\eta}\cdot\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)\cdot\boldsymbol{M}_{\eta\leftarrow\varepsilon}$ to $\varepsilon(a)$ gives $\varepsilon(\lambda(a))$. First, left-multiplying $\varepsilon(a)$ by $\boldsymbol{M}_{\eta\leftarrow\varepsilon}$ gives $\eta(a)$ by Lemma 1. Next, left-multiplying $\eta(a)$ by $\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)$ yields $\eta(\lambda(a))$ by definition of $\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)$, and finally, left-multiplying $\eta(\lambda(a))$ by $\boldsymbol{M}_{\varepsilon\leftarrow\eta}$ gives $\varepsilon(\lambda(a))$, again by Lemma 1. □

In particular, Proposition 3 shows that any $\mathbb{F}_p$-linear map $\lambda : \mathbb{F}_{p^r} \to \mathbb{F}_{p^r}$ can be applied seamlessly during an encoding switch. In other words, whenever such a switch is already performed, the linear map can be folded into it for free. A notable example is the Frobenius map $\mathsf{Pow}_\beta : a \mapsto a^{p^\beta}$ for $\beta \geqslant 0$, which is $\mathbb{F}_p$-linear.

*Example 1.* We will use the following as a running example. Suppose we want to compute a coefficient encoding of the product $a^p b$, for two elements $a, b \in \mathbb{F}_{p^r}$, starting from their coefficient encodings $\eta(a)$ and $\eta(b)$. This is achieved by the circuit shown in Figure 4.

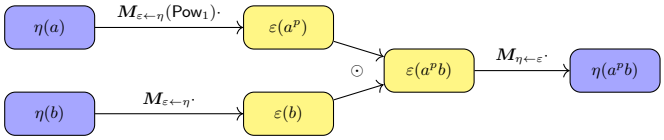


**Fig. 4.** Replicating the circuit $(a, b) \mapsto a^p b$ with $a, b \in \mathbb{F}_{p^r}$ starting and ending with coefficient encodings.

## 5 Finite-Field Arithmetic in CKKS

In this section, we explain how the above encoding techniques allow us to perform finite-field arithmetic within CKKS.

### 5.1 Packing and Encryption

Encrypting a single element $a \in \mathbb{F}_{p^r}$ via $\varepsilon$ requires $r$ complex slots. Since a CKKS ciphertext offers $N/2$ complex slots, we can therefore pack $k \leqslant \lfloor N/(2r) \rfloor$ elements of $\mathbb{F}_{p^r}$ into a single ciphertext. Let $n \in [\![1, N/2]\!]$ be the smallest power of two satisfying $n \geqslant rk$; we use $n$ as the effective number of complex slots in our ciphertexts. (While one could always take $n = N/2$, choosing a smaller $n$ whenever possible speeds up bootstrapping [14].) Given a vector $\boldsymbol{a} = (a_\ell)_{\ell \in [\![k]\!]} \in \mathbb{F}_{p^r}^k$, we arrange the $k$ spectral encodings $\varepsilon(a_\ell)$ and coefficient encodings $\eta(a_\ell)$ for $\ell \in [\![k]\!]$ into vectors of length $n$ as follows:

$$\varepsilon_{\mathsf{pack}}(\boldsymbol{a}) = (\varepsilon(a_0), \ldots, \varepsilon(a_{k-1}), \mathbf{0}) \in \mathbb{C}^n,$$
$$\eta_{\mathsf{pack}}(\boldsymbol{a}) = (\eta(a_0), \ldots, \eta(a_{k-1}), \mathbf{0}) \in \mathbb{Z}^n,$$

where $\mathbf{0}$ denotes the zero vector of length $n - kr$. The slot layout of $\varepsilon_{\mathsf{pack}}(\boldsymbol{a})$ is depicted in Figure 5. When $k = 1$, we will simply write $\varepsilon_{\mathsf{pack}}(a)$ and $\eta_{\mathsf{pack}}(a)$.

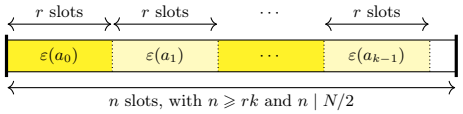


**Fig. 5.** Slot layout for spectral encoding $\varepsilon_{\mathsf{pack}}(\boldsymbol{a}) \in \mathbb{C}^n$ with $\boldsymbol{a} = (a_\ell)_{\ell \in [\![k]\!]} \in \mathbb{F}_{p^r}^k$.

The packed vectors $\varepsilon_{\mathsf{pack}}(\boldsymbol{a}) \in \mathbb{C}^n$ and $\eta_{\mathsf{pack}}(\boldsymbol{a}) \in \mathbb{Z}^n$ can then be mapped, via the standard CKKS encoding procedure, to plaintext polynomials $\mathsf{Ecd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a})) \in \mathcal{R}$ and $\mathsf{Ecd}(\eta_{\mathsf{pack}}(\boldsymbol{a})) \in \mathcal{R}$. Encrypting these plaintexts yields ciphertexts of the form $\mathbf{ct} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a})) \in \mathcal{R}_q^2$ or $\mathbf{ct} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a})) \in \mathcal{R}_q^2$, which we call a *spectral encryption* and a *coefficient encryption*, respectively. We will typically start from a coefficient encryption, as this representation is the one

most commonly used in prior work. In Proposition 5, we explain how to switch between spectral and coefficient encryptions homomorphically. Further, in Note 2, we explain why our setting favors the proposed *consecutive packing layout* over an alternative *interleaved packing layout*.

### 5.2 Basic Homomorphic Operations

We now summarize the two main homomorphic properties of spectral encryptions. The approximate symbol accounts for the usual CKKS approximation error, which is analyzed later in Section 6.

**Proposition 4.** *Consider two CKKS ciphertexts* $\mathbf{ct}_{\boldsymbol{a}} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a}))$ *and* $\mathbf{ct}_{\boldsymbol{b}} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{b}))$*, with* $\boldsymbol{a} = (a_\ell)_{\ell \in [\![k]\!]} \in \mathbb{F}_{p^r}^k$ *and* $\boldsymbol{b} = (b_\ell)_{\ell \in [\![k]\!]} \in \mathbb{F}_{p^r}^k$*. Then, the following properties hold:*

$$\mathbf{ct}_{\boldsymbol{a}} + \mathbf{ct}_{\boldsymbol{b}} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a} + \boldsymbol{b})), \qquad \mathbf{ct}_{\boldsymbol{a}} \times \mathbf{ct}_{\boldsymbol{b}} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a} \odot \boldsymbol{b})).$$

*Proof.* Both follow directly from Proposition 2, combined with the homomorphic properties of CKKS. For instance, we have:

$$\begin{aligned}
\mathbf{ct}_{\boldsymbol{a}} \times \mathbf{ct}_{\boldsymbol{b}} &\approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a}) \odot \varepsilon_{\mathsf{pack}}(\boldsymbol{b})) \\
&= \mathsf{EncEcd}(\varepsilon(a_0) \odot \varepsilon(b_0), \dots, \varepsilon(a_{k-1}) \odot \varepsilon(b_{k-1}), \mathbf{0}) \\
&= \mathsf{EncEcd}(\varepsilon(a_0 \cdot b_0), \dots, \varepsilon(a_{k-1} \cdot b_{k-1}), \mathbf{0}) \\
&= \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a} \odot \boldsymbol{b})),
\end{aligned}$$

and a similar computation shows the additive property. □

### 5.3 Homomorphic Multiplications by Matrices

In this section, we explain how to homomorphically switch between spectral and coefficient encryptions and apply $\mathbb{F}_p$-linear maps. Previously, we worked with square matrices $\boldsymbol{M}$ of size $r$ acting on a spectral encoding $\varepsilon(a)$ or a coefficient encoding $\eta(a)$ of a single field element $a \in \mathbb{F}_{p^r}$. To apply the same matrix simultaneously to each component of a packed vector $\boldsymbol{a} = (a_\ell)_{\ell \in [\![k]\!]} \in \mathbb{F}_{p^r}^k$, we extend $\boldsymbol{M}$ to a block-diagonal matrix. Specifically, for a complex matrix $\boldsymbol{M}$ of size $r$, we define its *extended version* by $\mathsf{Ext}(\boldsymbol{M}) = \mathsf{Diag}(\boldsymbol{M}, \dots, \boldsymbol{M}, \mathbf{0}) \in \mathbb{C}^{n \times n}$, where $\boldsymbol{M}$ is repeated $k$ times and $\mathbf{0}$ denotes the zero matrix of size $n - kr$. By construction, for any vector $\boldsymbol{z} = (\boldsymbol{z}_0, \dots, \boldsymbol{z}_{k-1}, \mathbf{0}) \in \mathbb{C}^n$ with $\boldsymbol{z}_\ell \in \mathbb{C}^r$ for every $\ell \in [\![k]\!]$, we have $\mathsf{Ext}(\boldsymbol{M}) \cdot \boldsymbol{z} = (\boldsymbol{M}\boldsymbol{z}_0, \dots, \boldsymbol{M}\boldsymbol{z}_{k-1}, \mathbf{0})$. Since we intend to apply these extended matrices homomorphically, we write $\mathsf{EcdExt}(\boldsymbol{M})$ for the extended and encoded matrix $\mathsf{Ecd}(\mathsf{Ext}(\boldsymbol{M}))$. Note that $\mathsf{DiagSupp}(\mathsf{Ext}(\boldsymbol{M})) \subseteq [\![n - r + 1, n - 1]\!] \cup [\![0, r - 1]\!]$, and hence it is contained in an interval modulo $n$ of length $2r - 1$. Therefore, we can apply $\mathsf{EcdExt}(\boldsymbol{M})$ homomorphically using $\mathcal{O}(r)$ plaintext–ciphertext multiplications and $\mathcal{O}(\sqrt{r})$ rotations.

The subsequent result shows how to homomorphically switch between encodings and apply $\mathbb{F}_p$-linear maps in the meantime. Approximation errors are again studied later in Section 6.

**Proposition 5.** *Consider a vector* $\boldsymbol{a} = (a_\ell)_{\ell \in [\![k]\!]} \in \mathbb{F}_{p^r}^k$ *of field elements and two ciphertexts* $\mathbf{ct}_\eta \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}))$ *and* $\mathbf{ct}_\varepsilon \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a}))$. *Let further* $\lambda : \mathbb{F}_{p^r} \to \mathbb{F}_{p^r}$ *be an* $\mathbb{F}_p$*-linear map. Then, we have:*

$$\begin{aligned}
\mathsf{EcdExt}(\boldsymbol{M}_{\varepsilon\leftarrow\eta}) \times \mathbf{ct}_\eta &\approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a})),\\
\mathsf{EcdExt}(\boldsymbol{M}_{\eta\leftarrow\varepsilon}) \times \mathbf{ct}_\varepsilon &\approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a})),\\
\mathsf{EcdExt}(\boldsymbol{M}_{\varepsilon\leftarrow\eta}(\lambda)) \times \mathbf{ct}_\eta &\approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\lambda(\boldsymbol{a}))),\\
\mathsf{EcdExt}(\boldsymbol{M}_{\eta\leftarrow\varepsilon}(\lambda)) \times \mathbf{ct}_\varepsilon &\approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\lambda(\boldsymbol{a}))),\\
\mathsf{EcdExt}(\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)) \times \mathbf{ct}_\eta &\approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\lambda(\boldsymbol{a}))),\\
\mathsf{EcdExt}(\boldsymbol{M}_{\varepsilon\leftarrow\varepsilon}(\lambda)) \times \mathbf{ct}_\varepsilon &\approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\lambda(\boldsymbol{a}))).
\end{aligned}$$

*Proof.* We prove the fourth relation; the other ones are analogous. By the homomorphic matrix-multiplication property, it is enough to verify the corresponding plaintext identity. By definition of the extended matrix, we have:

$$\begin{aligned}
\mathsf{Ext}(\boldsymbol{M}_{\eta\leftarrow\varepsilon}(\lambda)) \cdot \varepsilon_{\mathsf{pack}}(\boldsymbol{a}) &= (\boldsymbol{M}_{\eta\leftarrow\varepsilon}(\lambda) \cdot \varepsilon(a_0), \ldots, \boldsymbol{M}_{\eta\leftarrow\varepsilon}(\lambda) \cdot \varepsilon(a_{k-1}), \mathbf{0})\\
&= (\eta(\lambda(a_0)), \ldots, \eta(\lambda(a_{k-1})), \mathbf{0})\\
&= \eta_{\mathsf{pack}}(\lambda(\boldsymbol{a})),
\end{aligned}$$

where the second equality follows from Proposition 3. □

*Example 2.* Continuing Example 1, we can now homomorphically evaluate the circuit $(\boldsymbol{a}, \boldsymbol{b}) \mapsto \boldsymbol{a}^p \odot \boldsymbol{b}$ for vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_{p^r}^k$. Starting from coefficient encryptions $\mathbf{ct}_{\boldsymbol{a}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}))$ and $\mathbf{ct}_{\boldsymbol{b}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{b}))$, we get $\mathbf{ct}_{\mathsf{out}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}^p \odot \boldsymbol{b}))$ via $\mathbf{ct}_{\mathsf{out}} \leftarrow \mathsf{exampleCircuit}(\mathbf{ct}_{\boldsymbol{a}}, \mathbf{ct}_{\boldsymbol{b}})$ using Algorithm 1. This consumes three multiplicative levels in CKKS. Apart from the ciphertext–ciphertext multiplication, the circuit consists of three matrix–ciphertext multiplications. Each such multiplication uses $\mathcal{O}(r)$ plaintext–ciphertext multiplications and $\mathcal{O}(\sqrt{r})$ rotations. If the circuit is followed by bootstrapping, the final matrix–ciphertext multiplication can be folded into the bootstrapping procedure, saving one multiplicative level; see Section 7 for details.

**Algorithm 1:** exampleCircuit (evaluation of $(\boldsymbol{a}, \boldsymbol{b}) \mapsto \boldsymbol{a}^p \odot \boldsymbol{b}$ on coefficient encryptions).

**Input:** Ciphertext $\mathbf{ct}_{\boldsymbol{a}}$ ▷ $\mathbf{ct}_{\boldsymbol{a}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}))$
**Input:** Ciphertext $\mathbf{ct}_{\boldsymbol{b}}$ ▷ $\mathbf{ct}_{\boldsymbol{b}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{b}))$
**Output:** Ciphertext $\mathbf{ct}_{\mathsf{out}}$ ▷ $\mathbf{ct}_{\mathsf{out}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}^p \odot \boldsymbol{b}))$
1: $\mathbf{ct}'_{\boldsymbol{a}} \leftarrow \mathsf{EcdExt}(\boldsymbol{M}_{\varepsilon\leftarrow\eta}(\mathsf{Pow}_1)) \times \mathbf{ct}_{\boldsymbol{a}}$
2: $\mathbf{ct}'_{\boldsymbol{b}} \leftarrow \mathsf{EcdExt}(\boldsymbol{M}_{\varepsilon\leftarrow\eta}) \times \mathbf{ct}_{\boldsymbol{b}}$
3: $\mathbf{ct}_{\mathsf{out}} \leftarrow \mathbf{ct}'_{\boldsymbol{a}} \times \mathbf{ct}'_{\boldsymbol{b}}$
4: $\mathbf{ct}_{\mathsf{out}} \leftarrow \mathsf{EcdExt}(\boldsymbol{M}_{\eta\leftarrow\varepsilon}) \times \mathbf{ct}_{\mathsf{out}}$
5: **return** $\mathbf{ct}_{\mathsf{out}}$

*Note 2.* In Section 5.1, we use a *consecutive packing layout*, where the $k$ spectral encodings $\varepsilon(a_\ell)$ are concatenated. A natural alternative is an *interleaved packing layout*, which first stores all first entries of the $\varepsilon(a_\ell)$, then all second entries, and so on. For such a layout, given a matrix $\boldsymbol{M} = (M_{i,j})_{i,j\in[\![r]\!]}$ of size $r$, we would need to define $\mathsf{Ext}(\boldsymbol{M})$ by replacing each entry $M_{i,j}$ by the scaled identity block $M_{i,j}\boldsymbol{I}_k$, followed by zero-padding to size $n$. When $r$ is a power of two, this roughly halves the number of nonzero diagonals. However, $\mathsf{DiagSupp}(\mathsf{Ext}(\boldsymbol{M}))$ would no longer be contained in a short interval modulo $n$. Thus, when such a multiplication is followed by bootstrapping, we can no longer fold it into the first slot-to-coefficient layer, and hence cannot save the corresponding multiplicative level (see Section 7). We therefore use consecutive packing throughout.

## 6 Correctness-Management Techniques

The previous section shows how to realize arithmetic over $\mathbb{F}_{p^r}$ through homomorphic computations under CKKS. While this enables efficient SIMD-style evaluation, it also introduces nontrivial correctness challenges: CKKS is approximate and operates in characteristic zero, whereas computations in $\mathbb{F}_{p^r}$ are exact and performed modulo $p$.

Our construction represents field elements $a \in \mathbb{F}_{p^r} = \mathbb{F}_p[X]/(f)$ by lifts $\hat{a} \in \mathbb{Z}[X]/(F)$. Since CKKS arithmetic is approximate, homomorphic evaluation manipulates a perturbed element $\hat{a} + e$ rather than the exact lift $\hat{a}$, where $e \in \mathbb{C}[X]/(F)$ is an error polynomial. We call such an element $\hat{a}+e$ a *perturbed lift* of $a$. Correct decoding requires this error to stay small: the condition $\|e\|_\infty < 1/2$ ensures that coefficientwise rounding recovers $\hat{a}$. The coefficients of the lift $\hat{a}$ should also remain small: multiplication will scale the error by these coefficients.

Therefore, in this section, we analyze and control coefficient growth of both lifts and errors throughout the computation. We derive worst-case bounds, explain the typical-case growth behavior, give high-probability correctness guarantees, and explain how suitable choices of the polynomial $f$ lead to slower coefficient growth. In Section 7, we describe how bootstrapping can be used to reset the ciphertext by refreshing its modulus, lowering its noise, and symmetrically reducing lifted coefficients modulo $p$.

### 6.1 Worst-Case Lift and Error Bounds

We now provide worst-case bounds on the coefficient growth of both lifts and errors while replicating addition, multiplication, and application of $\mathbb{F}_p$-linear maps within $\mathbb{F}_{p^r}$ through operations in $\mathbb{C}[X]/(F)$.

The main source of growth is multiplication in the quotient ring: even if two representatives have small coefficients, their product may acquire large coefficients after reduction modulo $F$:

**Proposition 6.** *Let $F \in \mathbb{C}[X]$ be a monic polynomial of degree $r$. Then, for all $P, Q \in \mathbb{C}[X]/(F)$, we have $\|PQ\|_\infty \leqslant \mathcal{C}_F \cdot \|P\|_\infty \cdot \|Q\|_\infty$, where the constant $\mathcal{C}_F$ is defined as $\mathcal{C}_F = r \cdot (1 + \|F\|_\infty)^{r-1}$.*

We defer the proof to Appendix B. The constant $\mathcal{C}_F$ can later be replaced by a significantly smaller constant $\mathcal{C}'_F$ for the modulus polynomials used in practice.

In what follows, given an $\mathbb{F}_p$-linear map $\lambda : \mathbb{F}_{p^r} \to \mathbb{F}_{p^r}$, we denote by $\hat{\lambda} : \mathbb{C}[X]/(F) \to \mathbb{C}[X]/(F)$ the $\mathbb{C}$-linear map whose matrix with respect to the basis $1, X, \ldots, X^{r-1}$ is the previously introduced matrix $\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda) \in \mathbb{Z}^{r\times r}$. The procedure described in Section 4.3 for applying $\lambda$ then corresponds precisely to applying $\hat{\lambda}$ to elements of $\mathbb{C}[X]/(F)$.

**Proposition 7.** *Let $a, b \in \mathbb{F}_{p^r}$, and let $\hat{a} + e_a, \hat{b} + e_b \in \mathbb{C}[X]/(F)$ be perturbed lifts of $a$ and $b$, respectively. Assume that $\|\hat{a}\|_\infty, \|\hat{b}\|_\infty \leqslant B$ and $\|e_a\|_\infty, \|e_b\|_\infty \leqslant \delta$ for some bounds $B, \delta > 0$. Then, we have:*

$$\begin{aligned}(\hat{a} + e_a) + (\hat{b} + e_b) &= \widehat{a+b} + e_{\mathsf{add}},\\ (\hat{a} + e_a) \cdot (\hat{b} + e_b) &= \widehat{ab} + e_{\mathsf{mul}},\\ \hat{\lambda}(\hat{a} + e_a) &= \widehat{\lambda(a)} + e_\lambda,\end{aligned}$$

*where the terms on the right-hand side satisfy the bounds given in Table 2.*

**Table 2.** Worst-case lift and error bounds.

| Lift bound | Error bound |
|---|---|
| $\|\widehat{a+b}\|_\infty \leqslant 2B$ | $\|e_{\mathsf{add}}\|_\infty \leqslant 2\delta$ |
| $\|\widehat{ab}\|_\infty \leqslant \mathcal{C}_F B^2$ | $\|e_{\mathsf{mul}}\|_\infty \leqslant \mathcal{C}_F\left(2\delta B + \delta^2\right)$ |
| $\|\widehat{\lambda(a)}\|_\infty \leqslant \frac{1}{2}prB$ | $\|e_\lambda\|_\infty \leqslant \frac{1}{2}\delta pr$ |

*Proof.* For addition, we have $(\hat{a}+e_a)+(\hat{b}+e_b) = \widehat{a+b}+e_{\mathsf{add}}$, where $\widehat{a+b} = \hat{a}+\hat{b}$ and $e_{\mathsf{add}} = e_a + e_b$. The bounds then follow from sub-additivity of the infinity norm. For multiplication, we have $(\hat{a}+e_a)(\hat{b}+e_b) = \widehat{ab}+e_{\mathsf{mul}}$, where $\widehat{ab} = \hat{a}\hat{b}$ and $e_{\mathsf{mul}} = \hat{a}e_b + e_a\hat{b} + e_a e_b$. Applying Proposition 6 to each product term gives the stated bounds. Finally, for the linear map, we have $\hat{\lambda}(\hat{a} + e_a) = \widehat{\lambda(a)} + e_\lambda$, where $\widehat{\lambda(a)} = \hat{\lambda}(\hat{a})$ and $e_\lambda = \hat{\lambda}(e_a)$. Now recall that $\hat{\lambda}$ is represented by the matrix $\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)$. Thus, for every $P \in \mathbb{C}[X]/(F)$, each coefficient of $\hat{\lambda}(P)$ is a sum of at most $r$ coefficients of $P$ multiplied by entries of $\boldsymbol{M}_{\eta\leftarrow\eta}(\lambda)$. These entries lie in $\mathbb{Z}_p$, and hence have absolute value at most $p/2$. Therefore, $\|\hat{\lambda}(P)\|_\infty \leqslant (1/2)pr\|P\|_\infty$. Applying this with $P = \hat{a}$ and $P = e_a$ gives the stated bounds. □

Note that applying the encoding-switch matrices $\boldsymbol{M}_{\varepsilon\leftarrow\eta}$ and $\boldsymbol{M}_{\eta\leftarrow\varepsilon}$ does not affect the bounds from Proposition 7. Indeed, these matrices only change how a fixed element of $\mathbb{C}[X]/(F)$ is represented in the slots, by moving between coefficient and spectral encodings. They do not change the underlying polynomial, and hence do not change the coefficient norm tracked in the bounds above.

### 6.2 Choice of Modulus Polynomial

The multiplication bounds in Proposition 7 are governed by the constant $\mathcal{C}_F$ from Proposition 6. This constant is exponential in $r$, reflecting the possible coefficient blow-up caused by reduction modulo $F$. The situation improves substantially when $F$ contains no high-degree terms besides its leading monomial.

Let us say that a monic polynomial $F$ of degree $r$ is *truncatable* if $F - X^r$ has degree at most $\lceil r/2 \rceil$. In that case, reduction modulo $F$ can be controlled much more tightly, leading to the following improved product bound:

**Proposition 8.** *Let $F \in \mathbb{C}[X]$ be a truncatable monic polynomial of degree $r$, and let $H$ be the number of nonzero non-leading coefficients of $F$. Then, for all $P, Q \in \mathbb{C}[X]/(F)$, we have $\|PQ\|_\infty \leqslant \mathcal{C}'_F \cdot \|P\|_\infty \cdot \|Q\|_\infty$, where the constant $\mathcal{C}'_F$ is defined as $\mathcal{C}'_F = r\big(1 + H\|F\|_\infty + H^2\|F\|_\infty^2\big)$.*

*Proof.* Consider the natural projection $\pi : \mathbb{C}[X] \to \mathbb{C}[X]/(F)$, and let $c = \mathcal{C}'_F/r$. We first prove that for every polynomial $R \in \mathbb{C}[X]$ of degree at most $2r-2$, we have $\|\pi(R)\|_\infty \leqslant c \cdot \|R\|_\infty$. This is enough: taking $R = \hat{P}\hat{Q}$ for lifts $\hat{P}, \hat{Q} \in \mathbb{C}[X]$ of $P$ and $Q$ of degrees at most $r-1$ gives $\|R\|_\infty \leqslant r\|P\|_\infty\|Q\|_\infty$ (each coefficient of $R$ is a sum of at most $r$ products of one coefficient of $P$ with one coefficient of $Q$), and so $\|PQ\|_\infty = \|\pi(R)\|_\infty \leqslant c \cdot \|R\|_\infty \leqslant \mathcal{C}'_F \cdot \|P\|_\infty \cdot \|Q\|_\infty$.

Write $F = X^r + \sum_{u \in U} F_u X^u$, where $U \subseteq [\![\lceil r/2 \rceil + 1]\!]$ has cardinality $H$. For $w \in [\![r, 2r-2]\!]$, we have:

$$\begin{aligned}\pi(X^w) &= -\sum_{u \in U} F_u \pi(X^{u+w-r}) \\ &= -\sum_{\substack{u \in U \\ u+w-r<r}} F_u X^{u+w-r} + \sum_{\substack{u,v \in U \\ u+w-r \geqslant r}} F_u F_v X^{u+v+w-2r},\end{aligned}$$

where we used that $\pi\big(X^{u+v+w-2r}\big) = X^{u+v+w-2r}$ (since $u + v + w - 2r \leqslant \lceil r/2 \rceil + \lceil r/2 \rceil + (2r-2) - 2r < r$). Returning to $R = \sum_{w=0}^{2r-2} R_w X^w$, we then have:

$$\begin{aligned}\pi(R) &= \sum_{i=0}^{r-1} R_i X^i + \sum_{w=r}^{2r-2} R_w \pi(X^w) \\ &= \sum_{i=0}^{r-1} X^i \left( R_i - \sum_u R_{i+r-u} F_u + \sum_{u,v} R_{i+2r-u-v} F_u F_v \right),\end{aligned}$$

where, for each $i$, the sums over $u$ and $(u, v)$ are taken over suitable subsets of $U$ and $U^2$, respectively. It follows that each coefficient of $\pi(R)$ is bounded by $c \cdot \|R\|_\infty$, as required. □

We therefore choose the modulus polynomial $f \in \mathbb{F}_p[X]$ to be truncatable whenever possible. Its lift $F$ then satisfies the same property, and so the constant $\mathcal{C}_F$ in Table 2 can be replaced by the significantly smaller constant $\mathcal{C}'_F$ from Proposition 8.

Asymptotically, it seems that we can choose the modulus polynomial $f \in \mathbb{F}_p[X]$ so that its lift $F$ satisfies $\mathcal{C}'_F = \mathcal{O}(r)$. The following heuristic provides some intuition for why such choices should be plentiful:

**Heuristic 1.** *Let $H, K \geqslant 1$ be two integers with $K < p/2$. Then, as $r \to \infty$, there are $\Theta\big(r^{H-1}\big)$ monic irreducible polynomials $f \in \mathbb{F}_p[X]$ of degree $r$ that are truncatable, satisfy $\|f\|_\infty \leqslant K$, and have at most $H$ nonzero non-leading coefficients.*

*Explanation.* Ignoring irreducibility first, truncatability leaves only the coefficients of $X^0, \ldots, X^{\lceil r/2 \rceil}$ to be chosen. Since $H$ and $K$ are fixed, there are $\Theta\big(r^H\big)$ ways to choose up to $H$ positions among these $\lceil r/2 \rceil + 1$ monomials, and only $\mathcal{O}(1)$ ways to assign each chosen monomial a coefficient in $\{\pm 1, \ldots, \pm K\} \subseteq \mathbb{F}_p$. Finally, a random monic polynomial of degree $r$ over $\mathbb{F}_p$ is irreducible with probability about $1/r$. Thus, one expects $\Theta\big(r^{H-1}\big)$ irreducible choices for $f$. □

In particular, this heuristic suggests the existence of such polynomials $f$ for all sufficiently large $r$. We also verified existence computationally in the small-parameter regime relevant to our implementation. For $H = 4$ and $K = 2$, our search found such a polynomial $f \in \mathbb{F}_p[X]$ for $p$ and $r$ up to 6 and 7 bits, respectively. Together with the heuristic above, this suggests that one can generally choose $f$ so that its lift $F$ satisfies $\mathcal{C}'_F = \mathcal{O}(r)$.

### 6.3 Typical-Case Lift and Error Growth

The bounds from Proposition 7 correspond to a worst-case scenario in which all coefficient contributions add coherently. In practice, one instead expects substantial cancellation. Under the standard *random-cancellation heuristic*, we model these contributions as independent centered terms of comparable magnitude. Their variances therefore add, so a sum of $r$ such terms should typically grow like $\sqrt{r}$ rather than $r$. Applying this heuristic within the proofs of Proposition 7 and Proposition 8 leads to the typical-case growth estimates summarized in Table 3, where all objects are defined as in Proposition 7, and where the lift $F$ is assumed to satisfy $\mathcal{C}'_F = \mathcal{O}(r)$, as motivated in Section 6.2.

**Table 3.** Lift and error growth under random-cancellation heuristics.

| Lift growth | Error growth |
|---|---|
| $\|\widehat{a+b}\|_\infty = \mathcal{O}(B)$ | $\|e_{\mathsf{add}}\|_\infty = \mathcal{O}(\delta)$ |
| $\|\widehat{ab}\|_\infty = \mathcal{O}(B^2\sqrt{r})$ | $\|e_{\mathsf{mul}}\|_\infty = \mathcal{O}(\delta B\sqrt{r})$ |
| $\|\widehat{\lambda(a)}\|_\infty = \mathcal{O}(pB\sqrt{r})$ | $\|e_\lambda\|_\infty = \mathcal{O}(\delta p\sqrt{r})$ |

### 6.4 High-Probability Error Bounds

The previous sections analyzed the growth of lifts and errors in $\mathbb{C}[X]/(F)$ from two complementary viewpoints. The worst-case bounds provide rigorous guarantees that hold uniformly over all inputs and error configurations, while the typical-case estimates explain the substantially smaller growth expected in practice under random-cancellation heuristics. For concrete parameter selection, however, we need a middle ground: quantitative error bounds that are less pessimistic than worst-case estimates, but still imply correct decoding up to negligible failure probability.

Rather than deriving closed-form error formulas for arbitrary circuits, we describe a procedure for certifying correctness with high probability. The idea is to model slot errors probabilistically, estimate the resulting variance amplification, and convert this estimate into an upper bound on the decryption-failure probability. We illustrate this procedure on a concrete example below.

The basic correctness condition is simple: when the slots contain coefficient encodings, decoding succeeds as soon as every slot error has magnitude less than $1/2$. Thus, we aim to ensure that this condition fails only with negligible probability, which is also the relevant requirement for $\mathsf{IND\text{-}CPA^D}$ security [12].

For this, we adapt standard error assumptions from the CKKS literature [7,22]. We model slot errors as independent centered complex Gaussians and track how their variance propagates through the circuit. We assume that the additional error introduced by rescaling and key switching remains negligible compared to the error already present in the slots. Any circuit composed of homomorphic additions, multiplications, rotations and conjugations induces a map $g : \mathbb{C}^m \to \mathbb{C}^n$ on the plaintext slots. Since each of these operations is polynomial in the real and imaginary parts of the slots, the induced map $g$ is itself such a polynomial. The following proposition shows that, in the small-error regime relevant to CKKS, such circuits amplify the slot variance by at most a circuit-dependent constant factor:

**Proposition 9.** *Let $g : \mathbb{C}^m \to \mathbb{C}^n$ be a multivariate polynomial in the real and imaginary parts of the input slots. Let $\boldsymbol{z} \in \mathbb{C}^m$ be a random input vector sampled according to some distribution $\mathcal{D}$ on $\mathbb{C}^m$ with bounded support, and let $\boldsymbol{e} \in \mathbb{C}^m$ be an input error vector independent of $\boldsymbol{z}$ with independent slots following centered complex Gaussian distributions of variance at most $\sigma^2$. Then the slots of the output error vector $g(\boldsymbol{z}+\boldsymbol{e}) - g(\boldsymbol{z})$ have variance $\mathcal{O}(\sigma^2)$ as $\sigma \to 0$, where the hidden constant only depends on $g$ and $\mathcal{D}$.*

We prove this result in Appendix C. It implies the existence of a constant $\alpha = \alpha_{\mathcal{D}}(g)$ such that, for sufficiently small $\sigma$, every output slot of $g(\boldsymbol{z}+\boldsymbol{e}) - g(\boldsymbol{z})$ has variance at most $\alpha^2\sigma^2$. We call $\alpha$ the *noise-amplification factor*. Under the additional heuristic assumption that the output error remains approximately Gaussian, standard tail bounds give:

**Heuristic 2.** *Under the assumptions of Proposition 9, for every threshold $\tau > 0$, we have for sufficiently small $\sigma > 0$ that:*

$$\mathbb{P}(\|g(\boldsymbol{z}+\boldsymbol{e}) - g(\boldsymbol{z})\|_\infty \geqslant \tau) \leqslant n \exp\left(-\frac{\tau^2}{\alpha^2\sigma^2}\right).$$

*Explanation.* Denote the $i$-th output slot of $g(\boldsymbol{z})$ by $g_i(\boldsymbol{z})$ for every $i \in [\![n]\!]$, and let $\delta_i = g_i(\boldsymbol{z}+\boldsymbol{e}) - g_i(\boldsymbol{z})$. By definition of $\alpha$, each $\delta_i$ has variance at most $\alpha^2\sigma^2$. Modelling $\delta_i$ as a centered complex Gaussian with variance $v_i \leqslant \alpha^2\sigma^2$, we have that $|\delta_i|^2$ is exponentially distributed with mean $v_i$. Hence, for every $i \in [\![n]\!]$, we have:

$$\mathbb{P}(|\delta_i| \geqslant \tau) = \mathbb{P}\big(|\delta_i|^2 \geqslant \tau^2\big) = \exp\left(-\frac{\tau^2}{v_i}\right) \leqslant \exp\left(-\frac{\tau^2}{\alpha^2\sigma^2}\right).$$

Applying the union bound yields:

$$\mathbb{P}(\|g(\boldsymbol{z}+\boldsymbol{e}) - g(\boldsymbol{z})\|_\infty \geqslant \tau) \leqslant \sum_{i=0}^{n-1} \mathbb{P}(|\delta_i| \geqslant \tau) \leqslant n \cdot \exp\left(-\frac{\tau^2}{\alpha^2\sigma^2}\right). \qquad \square$$

*Example 3.* We illustrate the use of the above framework on our running example, see Example 1 and Example 2. To recall, we start with ciphertexts $\mathbf{ct}_a \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(a))$ and $\mathbf{ct}_b \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(b))$ for $a, b \in \mathbb{F}_{p^r}$ (we assume that we encrypt only $k = 1$ field element per ciphertext for now). Algorithm 1 induces a map $g : \mathbb{C}^{2r} \to \mathbb{C}^r$ given by:

$$g(\boldsymbol{z}, \boldsymbol{w}) = \boldsymbol{M}_{\eta\leftarrow\varepsilon} \cdot ((\boldsymbol{M}_{\varepsilon\leftarrow\eta}(\mathsf{Pow}_1) \cdot \boldsymbol{z}) \odot (\boldsymbol{M}_{\varepsilon\leftarrow\eta} \cdot \boldsymbol{w})).$$

We sample the inputs $(\boldsymbol{z}, \boldsymbol{w})$ uniformly from $\mathbb{Z}_p^{2r}$, corresponding to symmetrically reduced coefficient encodings such as those obtained after bootstrapping. We then estimate the amplification factor $\alpha$ empirically by Monte-Carlo simulation. For instance, for $(p, r) = (5, 16)$ and truncatable modulus polynomial $f = X^{16}+X^3-1$, we obtain an estimated amplification factor $\alpha \approx 27$. Therefore, by Heuristic 2, if the input slots have error variance at most $\sigma^2$, then:

$$\mathbb{P}(\|g(\boldsymbol{z}+\boldsymbol{e}) - g(\boldsymbol{z})\|_\infty \geqslant 260\sigma) \leqslant 2^{-128}.$$

If we more generally pack $k$ field elements per ciphertext, Algorithm 1 induces a map $\tilde{g} : \mathbb{C}^{2kr} \to \mathbb{C}^{kr}$ in which $g$ is applied blockwise, while the amplification factor remains unchanged (since the circuit is simply repeated independently on each packed block). For the same parameters as above and $k = 2^{10}$, we obtain:

$$\mathbb{P}(\|\tilde{g}(\boldsymbol{z}+\boldsymbol{e}) - \tilde{g}(\boldsymbol{z})\|_\infty \geqslant 270\sigma) \leqslant 2^{-128}.$$

Therefore, if the input ciphertexts to Algorithm 1 contain coefficient encodings in their slots with error variance at most $\sigma^2$, and if $270\sigma < 1/2$, then the output ciphertext decrypts correctly with probability at least $1 - 2^{-128}$.

## 7 Bootstrapping

We now explain how to refresh ciphertexts during finite-field computations within CKKS. We start with a coefficient encryption of the form $\mathbf{ct} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}) + \boldsymbol{e}) \in \mathcal{R}_q^2$ at a small modulus $q$, where $\boldsymbol{a} \in \mathbb{F}_{p^r}^k$ and where the slot error satisfies $\|\boldsymbol{e}\|_\infty < 1/2$. At this stage of the computation, both the accumulated error and the integer entries of $\eta_{\mathsf{pack}}(\boldsymbol{a}) \in \mathbb{Z}^n$ may already be large. Since these integer entries are only interpreted modulo $p$, we ultimately only care about their centered representatives in $\mathbb{Z}_p$. As shown in Figure 6, bootstrapping addresses all the above issues simultaneously by:

- Refreshing the ciphertext modulus from $q$ to a larger modulus $\hat{q}$;
- Reducing the accumulated error from $\boldsymbol{e}$ to $\hat{\boldsymbol{e}}$ with $\|\hat{\boldsymbol{e}}\|_\infty = \mathcal{O}\big(\|\boldsymbol{e}\|_\infty^2\big)$;
- Re-centering lifted coefficients by reducing them symmetrically modulo $p$, *i.e.*, mapping $\eta_{\mathsf{pack}}(\boldsymbol{a}) \mapsto \eta_{\mathsf{pack}}(\boldsymbol{a}) \bmod p$.

The output will then be a ciphertext $\widehat{\mathbf{ct}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}) \bmod p + \hat{\boldsymbol{e}}) \in \mathcal{R}_{\hat{q}}^2$.

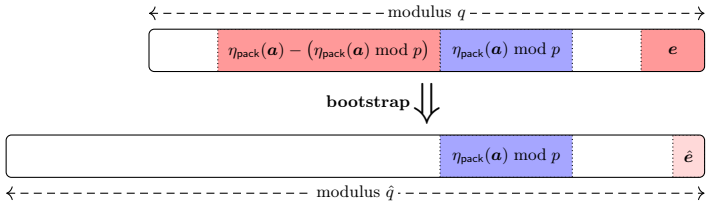


**Fig. 6.** Bootstrapping a ciphertext containing $\eta_{\mathsf{pack}}(\boldsymbol{a}) + \boldsymbol{e}$ in the slots.

**Coefficient Bootstrapping.** We build on the small-integer bootstrapping method SI-BTS [4], which fits our setting particularly well. At a high level, SI-BTS takes as input a ciphertext encrypting approximate integers and outputs a refreshed ciphertext at a higher modulus $\hat{q}$ encrypting these integers reduced modulo $p$, while also reducing the error quadratically. The procedure begins with a slot-to-coefficient operation, S2C [14]. A key observation for our setting is that the reduction modulo $p$ is already implicit in this step: by choosing the scaling factor incorporated into S2C appropriately, only the least significant $p$-digit of each coefficient is retained after S2C. While this phenomenon is already present in [4], it is made more explicit in subsequent works such as [39,44]. No further adjustments to SI-BTS are needed.

**Spectral Bootstrapping.** The original SI-BTS algorithm directly supports coefficient encryptions. However, it can also bootstrap spectral encryptions $\mathbf{ct} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a}))$ without consuming an additional multiplicative level. The idea is to fold the multiplication by the encoding-switch matrix $\mathsf{Ext}(\boldsymbol{M}_{\eta \leftarrow \varepsilon})$ (or,

if desired, by $\mathsf{Ext}(\boldsymbol{M}_{\eta\leftarrow\varepsilon}(\lambda))$ for some $\mathbb{F}_p$-linear map $\lambda : \mathbb{F}_{p^r} \to \mathbb{F}_{p^r}$) directly into the initial S2C step. Since S2C is itself linear, this folding simply amounts to composing two matrices into one. In practice, the S2C transform is decomposed into several layers, and the folding must therefore be integrated into the first of these layers; we defer the details to Appendix D. The resulting procedure outputs a coefficient encryption, therefore containing $\eta_{\mathsf{pack}}(\boldsymbol{a}) \bmod p$ in the slots. If a spectral encryption is desired afterwards, one final encoding switch is required.

We emphasize that this folding technique is not specific to the encoding-switch matrix. It applies to any matrix $\boldsymbol{M} \in \mathbb{C}^{n\times n}$, but is most efficient when the diagonal support of $\boldsymbol{M}$ is contained in a short interval modulo $n$. In that case, folding $\boldsymbol{M}$ into the first S2C layer is as efficient as applying $\boldsymbol{M}$ immediately before S2C, while saving one multiplicative level. In particular, this does not require the extension degree $r$ to be a power of two.

**Complex Packing Optimization.** We finally describe a throughput optimization for bootstrapping. First, note that SI-BTS can also bootstrap ciphertexts containing approximate integers simultaneously in the real and imaginary parts of the slots. As a result, the expensive S2C and C2S operations within SI-BTS can be shared by both parts, improving throughput. Some subroutines of SI-BTS still need to be applied separately to the two parts, such as IntRootBoot [4].

In our setting, given two coefficient encryptions $\mathbf{ct}_{\boldsymbol{a}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}))$ and $\mathbf{ct}_{\boldsymbol{b}} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{b}))$, we combine them into a single ciphertext $\mathbf{ct} \approx \mathsf{EncEcd}(\eta_{\mathsf{pack}}(\boldsymbol{a}) + \mathrm{i}\cdot\eta_{\mathsf{pack}}(\boldsymbol{b}))$ by computing $\mathbf{ct} \leftarrow \mathbf{ct}_{\boldsymbol{a}} + X^{N/2}\cdot\mathbf{ct}_{\boldsymbol{b}}$. We then bootstrap both parts simultaneously using the above variant of SI-BTS. The refreshed coefficient encryptions of $\boldsymbol{a}$ and $\boldsymbol{b}$ are recovered afterwards by taking real and imaginary parts, respectively.

The same idea applies to spectral encryptions. Given $\mathbf{ct}_{\boldsymbol{a}} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a}))$ and $\mathbf{ct}_{\boldsymbol{b}} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{b}))$, we first combine them into $\mathbf{ct} \approx \mathsf{EncEcd}(\varepsilon_{\mathsf{pack}}(\boldsymbol{a}) + \mathrm{i}\cdot\varepsilon_{\mathsf{pack}}(\boldsymbol{b}))$, and then apply the bootstrapping procedure with the folding technique described above. This is valid because for a matrix $\boldsymbol{M} \in \mathbb{C}^{n\times n}$ and vectors $\boldsymbol{z}, \boldsymbol{w} \in \mathbb{C}^n$, we have $\boldsymbol{M}(\boldsymbol{z} + \mathrm{i}\cdot\boldsymbol{w}) = \boldsymbol{M}\boldsymbol{z} + \mathrm{i}\cdot\boldsymbol{M}\boldsymbol{w}$. Hence, combining ciphertexts and then switching encodings is equivalent to first switching encodings separately and then combining the ciphertexts afterwards. As before, coefficient encryptions are recovered by taking real and imaginary parts. If spectral encryptions are desired instead, one finally applies the encoding switch to each part separately.

## 8 BGV over Extension Fields

In this section, we briefly discuss the BGV bootstrapping method used in our benchmarks and explain why certain pairs $(p, r)$ force BGV into wasteful packing regimes for arithmetic over $\mathbb{F}_{p^r}$.

**Instantiating BGV.** To support arithmetic over $\mathbb{F}_{p^r}$ in BGV, one chooses a cyclotomic index $m \geqslant 1$ such that $d = \mathrm{ord}_m(p)$ is a multiple of $r$, say $d = ur$.

The resulting RLWE ring has dimension $\phi(m)$, and each ciphertext packs $\phi(m)/d$ elements of $\mathbb{F}_{p^d}$. Since $\mathbb{F}_{p^r}$ embeds into $\mathbb{F}_{p^d}$ ($r$ divides $d$), the same packing capacity is available for arithmetic over $\mathbb{F}_{p^r}$. The ideal situation is therefore $d = r$. If instead $d = ur$ with $u \geqslant 2$, then only a fraction $1/u$ of the available plaintext space is used for $\mathbb{F}_{p^r}$-elements. We refer to this situation as a *wasteful packing regime*, and call $u$ the *waste factor*.

**Bootstrapping BGV.** We now briefly discuss the bootstrapping techniques considered for our benchmarks. As explained in [33], BGV admits a faster variant known as *thin bootstrapping*. However, this method only applies when the plaintext space is a prime field $\mathbb{F}_p$, and therefore does not support arithmetic over extension fields $\mathbb{F}_{p^r}$. Another relevant approach is [40], which leverages CKKS techniques to improve finite-field computations in BFV. However, this method also comes with restrictions. Since it relies on conventional CKKS, the cyclotomic index $m$ must be a power of two. Consequently, $p$ must be odd, since $p$ must be invertible modulo $m$. Moreover, note that $d = \mathrm{ord}_m(p)$ necessarily divides $|(\mathbb{Z}/m\mathbb{Z})^\times| = \phi(m) = m/2$ (by Lagrange's theorem from group theory), which is itself a power of two. Hence, this approach only supports extension degrees $r$ that are powers of two. To use a single BGV bootstrapping method that applies uniformly across all extension fields considered, we therefore choose the *thick bootstrapping* method of [33] as the BGV baseline in our benchmarks. Although HElib may no longer be the most heavily optimized implementation, it remains competitive with more recent libraries: for instance, Fheanor takes about $0.9\times$ to $1.4\times$ the time of HElib for thin bootstrapping, as reported in [45].

**Obstructions to Optimal Packing in BGV.** In thick bootstrapping, the dominant cost grows roughly linearly in $d$ due to $d$-fold digit extraction [33], while the number of packed field elements scales as $\phi(m)/d$. Consequently, if $d = ur$, then a wasteful packing regime typically incurs an asymptotic slowdown of $\Theta(u^2)$ compared to the ideal case $d = r$, and can therefore significantly degrade performance. However, for some pairs $(p, r)$, achieving the ideal situation $d = r$ already forces the ring dimension $\phi(m)$ to become impractically large. The following result yields one obstruction to optimal packing in BGV for some pairs $(p, r)$:

**Proposition 10.** *Let $p$ be prime and let $r$ be a prime power. Let $m \geqslant 1$ be such that $\mathrm{ord}_m(p) = r$. Then:*

$$\phi(m) \geqslant \min\{\phi(q) : q \text{ is a prime power and } \mathrm{ord}_q(p) = r\}.$$

*Proof.* Write the prime-power decomposition of $m$ as $m = \prod_i q_i$. The factors $q_i$ are pairwise coprime, so we have $r = \mathrm{ord}_m(p) = \mathrm{lcm}_i \mathrm{ord}_{q_i}(p)$. As $r$ is a prime power, one of the orders $\mathrm{ord}_{q_i}(p)$ must already equal $r$; say $\mathrm{ord}_{q_j}(p) = r$. Since $q_j$ divides $m$, we have that $\phi(q_j)$ divides $\phi(m)$, and therefore that $\phi(m) \geqslant \phi(q_j)$. The claim follows. □

For several parameter choices $(p, r)$ relevant in practice, the lower bound from Proposition 10 already forces $\phi(m)$ far beyond practical RLWE dimensions. Table 4 gives several concrete examples. We include the case $(p, r) = (3, 64)$ to illustrate that the obstruction is not always prohibitive. In practice, one may still choose parameters satisfying $\mathrm{ord}_m(p) = ur$ for some small integer $u \geqslant 2$. However, this necessarily introduces a wasteful packing regime and the associated performance degradation.

**Table 4.** Lower bounds on $\phi(m)$ for achieving $\mathrm{ord}_m(p) = r$.

| $p$ | $r$ | $\phi(m)$ lower bound |
|---|---|---|
| 2 | 127 | $1.70 \cdot 10^{38}$ |
| 2 | 128 | 274 176 |
| 2 | 256 | $5.97 \cdot 10^{16}$ |
| 3 | 13 | 797 160 |
| 3 | 64 | 128 |
| 5 | 11 | 12 207 030 |
| 11 | 27 | $5.56 \cdot 10^{18}$ |

## 9 Practical Results

We implemented our framework in the Lattigo library [50] on a MacBook Pro equipped with an Apple M4 processor (14 CPU cores and 24 GB of RAM), running macOS Sequoia 15.7.3.[2] We now compare our approach with BGV for bootstrapping finite-field elements and demonstrate that the proposed method is practically competitive across several parameter regimes.

**Evaluation Methodology.** In both CKKS and BGV, the dominant cost of finite-field arithmetic is the bootstrapping operation, which is required after a certain number of multiplications. Accordingly, we benchmark the bootstrapping procedure itself and compare the amortized cost per field element and per usable multiplicative level. The amortized time is defined as the total bootstrapping time divided by both the number of field elements packed into a ciphertext and the number of ciphertext–ciphertext multiplications that can be performed before the next bootstrap. All bootstrapping times are measured in single-threaded mode and averaged over 25 runs. For our method, we benchmark the spectral-to-spectral variant of the bootstrapping procedure, since spectral encryptions are the representation supporting native finite-field multiplication.

**Benchmark Setup.** For CKKS, we use ring dimensions $N = 2^{15}$ and $N = 2^{16}$. The bit sizes of the primes in the RNS chain [41] are given in Table 5. There, the

[2] The implementation and scripts used to run the benchmarks and reproduce the results of this section will be made available upon publication.

row "Usable" denotes the modulus consumed outside of bootstrapping. The row "SI-BTS without S2C/C2S" denotes the moduli used by the SI-BTS procedure itself, excluding those used for the S2C and C2S transformations. Finally, $P$ denotes the auxiliary moduli used for key switching. Thus, key-switching keys are provided at modulus $PQ$, whose bit size is reported in the last row.

**Table 5.** Bit-size contributions to the CKKS modulus chain for the benchmarked parameter sets.

| Modulus contribution | $N = 2^{15}$ | $N = 2^{16}$ |
|---|---|---|
| Base | 42 | 52 |
| S2C with $\boldsymbol{M}_{\eta\leftarrow\varepsilon}$ | $3 \cdot 42$ | $4 \cdot 42$ |
| Usable | $2 \cdot 42$ | 52 |
| $\boldsymbol{M}_{\varepsilon\leftarrow\eta}$ | 42 | 52 |
| SI-BTS without S2C/C2S | $8 \cdot 42$ | $15 \cdot 52$ |
| C2S | $3 \cdot 29$ | $3 \cdot 48$ |
| $P$ | $3 \cdot 42$ | $5 \cdot 52$ |
| $\log_2(PQ)$ | 843 | 1 508 |

The secret-key Hamming weight is chosen as $h = 256$ for $N = 2^{15}$ and $h = 192$ for $N = 2^{16}$, and the ephemeral secret-key Hamming weight [8] is chosen as $\tilde{h} = 32$. These parameters provide 128-bit security according to [1]. The only field-specific parameter in our framework is the modulus polynomial $f \in \mathbb{F}_p[X]$, which we always choose to be truncatable. Among such polynomials, we select instances yielding small noise-amplification factors $\alpha$ for multiplication. The resulting choices are summarized in Table 6. Using the error analysis from Section 6.4, these parameters ensure decryption-failure probability of at most $2^{-128}$, even when evaluating as many multiplications as permitted by the available usable levels before the next bootstrap.

**Table 6.** Chosen modulus polynomial $f \in \mathbb{F}_p[X]$ and resulting amplification factor $\alpha$.

| $p^r$ | $f$ | $\alpha$ |
|---|---|---|
| $2^8$ | $X^8 + X^4 + X^3 + X + 1$ | 6.0 |
| $2^{128}$ | $X^{128} + X^{63} + X^{38} + X + 1$ | 23 |
| $31^3$ | $X^3 - X + 1$ | 29 |

For BGV, we used the parameter-generation tool provided by HElib.[3] All parameter sets target 128-bit security; the only exception is the BGV parameter set for $(p, r) = (2, 128)$, which provides an estimated 126 bits of security. When

[3] Specifically, we used `misc/algen/algen.py` and `misc/params1a.cpp` from the HElib repository [34].

several parameter sets lead to a similar $\phi(m)$ and the same value for $\mathrm{ord}_m(p)$, we select the one with the highest throughput for bootstrapping.

**Results and Discussion.** The benchmark results are reported in Table 7. We selected parameter pairs $(p, r)$ that arise in concrete applications. The pair $(2, 8)$ is relevant for AES transciphering [23], while $(2, 128)$ is motivated by binary-field SNARKs, which use towers up to $\mathbb{F}_{2^{128}}$ [24]. Finally, the field $\mathbb{F}_{31^3}$ appears in concrete parameter sets for multivariate signature schemes that are not prone to the recent attack targeting binary-field-based UOV [47].

We observe that our CKKS-based approach substantially outperforms BGV for the binary-field instances. One reason is that SI-BTS is particularly efficient for small moduli such as $p = 2$. Another reason is that BGV is forced into wasteful packing regimes for both binary-field parameter sets.

For $(p, r) = (2, 8)$, the condition $\mathrm{ord}_m(p) = r$ implies that $m$ divides $2^r - 1 = 255$, which is too small to provide a secure RLWE dimension. We therefore choose parameters with $\mathrm{ord}_m(p) = 2r$, corresponding to a waste factor $u = 2$ and an expected slowdown of roughly $u^2 = 4$ compared to the ideal case $u = 1$. For $(p, r) = (2, 128)$, Proposition 10 already rules out practical choices with $\mathrm{ord}_m(p) \in \{r, 2r\}$. We therefore use parameters satisfying $\mathrm{ord}_m(p) = 3r$, yielding a waste factor $u = 3$ and an expected slowdown of about $9\times$. Thick bootstrapping in BGV then requires $3r = 384$ digit-extraction steps, which is the main source of the observed slowdown.

In contrast, the instance $(p, r) = (31, 3)$ is comparatively favorable to BGV, despite also having a waste factor of $u = 2$. Indeed, the value $u = 1$ would imply $\mathrm{ord}_m(p) = r = 3$, and hence that $m$ divides $p^r - 1 = 31^3 - 1 = 29\,790$. Thus, $\phi(m)$ divides $\phi(29\,790) = 7\,920$, a value too small to allow for bootstrapping. However, the waste factor is less harmful here because the total number of digit-extraction steps is small: only $ur = 2 \cdot 3 = 6$ are needed, which remains relatively cheap. At the same time, our method becomes less favorable, since SI-BTS is significantly more expensive for larger plaintext moduli such as $p = 31$. This also reduces the number of usable levels available in our CKKS setting.

All in all, the results suggest that our approach is particularly competitive for small characteristic $p$ and large extension degree $r$, whereas BGV remains competitive for larger characteristic and smaller extension degree.

## 10 Conclusion

We introduced a CKKS-based framework for arithmetic over finite fields $\mathbb{F}_{p^r}$, based on two complementary representations: a spectral encoding for native field arithmetic and a coefficient encoding for arithmetic in the slot algebra $\mathbb{F}_p^r$. Homomorphic switching between these representations allows computations to exploit both structures efficiently.

The resulting implementation in Lattigo shows that the approach is practical, achieving substantial improvements over BGV in amortized multiplication time

**Table 7.** Amortized bootstrapping time (in milliseconds) per packed field element and usable multiplicative level.

| $p^r$ | $2^8$ [23] | | $2^{128}$ [24] | | $31^3$ [47] | |
|---|---|---|---|---|---|---|
| Scheme | BGV | CKKS | BGV | CKKS | BGV | CKKS |
| RLWE dim. | $\phi(65\,535)$ $= 32\,768$ | $2^{15}$ | $\phi(49\,985)$ $= 36\,864$ | $2^{15}$ | $\phi(81\,095)$ $= 55\,440$ | $2^{16}$ |
| Field elem. | 2 048 | 4 096 | 96 | 256 | 9 240 | 21 844 |
| Usable levels | **6** | 2 | **19** | 2 | **18** | 1 |
| Time (s) | 146 | **2.60** | 4 505 | **7.11** | 179 | **13.9** |
| Amortized time (ms) | 11.9 | **0.317** | 2 470 | **13.9** | 1.07 | **0.636** |
| Speedup | 1.0× | **38**× | 1.0× | **178**× | 1.0× | **1.7**× |

with bootstrapping, especially for small characteristic $p$ and large extension degree $r$, while also providing substantially lower latency.

**Acknowledgments.** The first author acknowledges the support of the Luxembourgish FNR (Fonds National de la Recherche) through an Individual Grant (reference number: 17936291). The authors thank Jean-Sébastien Coron for his insightful comments and Robin Geelen for a helpful discussion on parameter selection for BGV in HElib.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Albrecht, M., Player, R., Scott, S.: On the Concrete Hardness of Learning with Errors. Journal of Mathematical Cryptology **9** (2015)
2. Alexandru, A., Kim, A., Polyakov, Y.: General Functional Bootstrapping Using CKKS. In: Advances in Cryptology – CRYPTO 2025. Springer (2025)
3. Bae, Y., Cheon, J.H., Kim, J., Stehlé, D.: Bootstrapping Bits with CKKS. In: Advances in Cryptology – EUROCRYPT 2024. Springer (2024)
4. Bae, Y., Kim, J., Stehlé, D., Suvanto, E.: Bootstrapping Small Integers with CKKS. In: Advances in Cryptology – ASIACRYPT 2024. Springer (2025)
5. Beullens, W.: MAYO: Practical Post-Quantum Signatures from Oil-and-Vinegar Maps. In: Selected Areas in Cryptography – SAC 2022. Springer (2022)
6. Boneh, D., Kim, J.: A Universal Blinder: One-round Blind Signatures from FHE (2026), `https://eprint.iacr.org/2026/574`
7. Bossuat, J.P., Costache, A., Mouchet, C., Nürnberger, L., Troncoso-Pastoriza, J.: Accurate and Composable Noise Estimates for CKKS with Application to Exact HE Computation. IACR Communications in Cryptology **2** (2025)
8. Bossuat, J.P., Troncoso-Pastoriza, J., Hubaux, J.P.: Bootstrapping for Approximate Homomorphic Encryption with Negligible Failure-Probability by Using Sparse-Secret Encapsulation. In: Applied Cryptography and Network Security. Springer (2022)
9. Brakerski, Z.: Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In: Advances in Cryptology – CRYPTO 2012. Springer (2012)

10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping. In: ITCS 2012: Innovations in Theoretical Computer Science. ACM (2012)
11. Chen, H., Chillotti, I., Song, Y.: Improved Bootstrapping for Approximate Homomorphic Encryption. In: Advances in Cryptology – EUROCRYPT 2019. Springer (2019)
12. Cheon, J.H., Choe, H., Passelègue, A., Stehlé, D., Suvanto, E.: Attacks Against the IND-CPA-D Security of Exact FHE Schemes. In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security. ACM (2024)
13. Cheon, J.H., Han, K., Hhan, M.: Improved Homomorphic Discrete Fourier Transforms and FHE Bootstrapping. IEEE Access **7** (2019)
14. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for Approximate Homomorphic Encryption. In: Advances in Cryptology – EUROCRYPT 2018. Springer (2018)
15. Cheon, J.H., Hanrot, G., Kim, J., Stehlé, D.: SHIP: A Shallow and Highly Parallelizable CKKS Bootstrapping Algorithm. In: Advances in Cryptology – EUROCRYPT 2025. Springer (2025)
16. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic Encryption for Approximate Numbers. In: Advances in Cryptology – ASIACRYPT 2017. Springer (2017)
17. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In: Advances in Cryptology – ASIACRYPT 2016. Springer (2016)
18. Choe, H., Kim, J., Stehlé, D., Suvanto, E.: Leveraging Discrete CKKS to Bootstrap in High Precision. In: Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security. ACM (2025)
19. Cooley, J.W., Tukey, J.: An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation **19** (1965)
20. Coron, J.S., Köstler, R.: Low-Latency Bootstrapping for CKKS using Roots of Unity (2025), `https://eprint.iacr.org/2025/651`
21. Coron, J.S., Seuré, T.: PaCo: Bootstrapping for CKKS via Partial CoeffToSlot. In: Advances in Cryptology – ASIACRYPT 2025. Springer (2026)
22. Costache, A., Curtis, B.R., Hales, E., Murphy, S., Ogilvie, T., Player, R.: On the Precision Loss in Approximate Homomorphic Encryption. In: Selected Areas in Cryptography – SAC 2023. Springer (2024)
23. Daemen, J., Rijmen, V.: The Design of Rijndael: AES – The Advanced Encryption Standard. Springer (2002)
24. Diamond, B.E., Posen, J.: Succinct Arguments over Towers of Binary Fields. In: Advances in Cryptology – EUROCRYPT 2025. Springer (2025)
25. Drucker, N., Moshkowich, G., Pelleg, T., Shaul, H.: BLEACH: Cleaning Errors in Discrete Computations over CKKS. Journal of Cryptology **37** (2024)
26. Ducas, L., Micciancio, D.: FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In: Advances in Cryptology – EUROCRYPT 2015. Springer (2015)
27. Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption (2012), `https://eprint.iacr.org/2012/144`
28. Gama, M., Heydari Beni, E., Kang, J., Spiessens, J., Vercauteren, F.: Blind zk-SNARKs For Private Proof Delegation and Verifiable Computation over Encrypted Data. IACR Communications in Cryptology **2** (2025)
29. Gao, M., Zheng, H.: FHE for SIMD Arithmetic Logic Units with Amortized $O(1)$ Bootstrapping per Ciphertext (2026), `https://eprint.iacr.org/2026/233`
30. Geelen, R., Vercauteren, F.: Fully Homomorphic Encryption for Cyclotomic Prime Moduli. In: Advances in Cryptology – EUROCRYPT 2025. Springer (2025)

31. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proceedings of the Annual ACM Symposium on Theory of Computing. ACM (2009)
32. Halevi, S., Shoup, V.: Algorithms in HElib. In: Advances in Cryptology – CRYPTO 2014. Springer (2014)
33. Halevi, S., Shoup, V.: Bootstrapping for HElib. Journal of Cryptology **34** (2021)
34. HElib developers: HElib (2023), `https://github.com/homenc/HElib`
35. Ju, J.H., Park, J., Kim, J., Kang, M., Kim, D., Cheon, J.H., Ahn, J.H.: NeuJeans: Private Neural Network Inference with Joint Optimization of Convolution and FHE Bootstrapping. In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security. ACM (2024)
36. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., Yoo, D.: General Bootstrapping Approach for RLWE-Based Homomorphic Encryption. IEEE Transactions on Computers **73** (2024)
37. Kim, J.: Efficient Homomorphic Integer Computer from CKKS. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025** (2025)
38. Kim, J.: Faster Homomorphic Integer Computer (2025), `https://eprint.iacr.org/2025/1440`
39. Kim, J., Noh, T.: Modular Reduction in CKKS. IACR Communications in Cryptology **2** (2025)
40. Kim, J., Seo, J., Song, Y.: Simpler and Faster BFV Bootstrapping for Arbitrary Plaintext Modulus from CKKS. In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security. ACM (2024)
41. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In: Advances in Cryptology – EUROCRYPT 2021. Springer (2021)
42. Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings. In: Advances in Cryptology – EUROCRYPT 2010. Springer (2010)
43. National Institute of Standards and Technology: Advanced Encryption Standard (AES) (2001), `https://doi.org/10.6028/NIST.FIPS.197`
44. Niu, C., Huang, Z., Yang, Z., Chen, Y., Kong, L., Hong, C., Wei, T.: XBOOT: Free-XOR Gates for CKKS with Applications to Transciphering. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025** (2025)
45. Okada, H., Player, R., Pohmann, S.: Fheanor: A New, Modular FHE Library for Designing and Optimising Schemes (2025), `https://eprint.iacr.org/2025/864`
46. Park, J.H., Passelègue, A., Stehlé, D.: THED: Threshold Dilithium from FHE (2026), `https://eprint.iacr.org/2026/638`
47. Ran, L.: Wedges, Oil, and Vinegar: An Analysis of UOV in the Exterior Algebra. In: Advances in Cryptology – EUROCRYPT 2026. Springer (2026)
48. Smart, N.P., Vercauteren, F.: Fully Homomorphic SIMD Operations. Designs, Codes and Cryptography **71** (2014)
49. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient Public Key Encryption Based on Ideal Lattices. In: Advances in Cryptology – ASIACRYPT 2009. Springer (2009)
50. Tune Insight: Lattigo v6 (2024), `https://github.com/tuneinsight/lattigo`

## A Distinct Roots of Polynomial Lifts

The goal of this section is to prove the following result:

**Proposition 1.** *Let $f \in \mathbb{F}_p[X]$ be an irreducible polynomial. Then, any lift $F \in \mathbb{Z}[X]$ of $f$ with the same degree as $f$ has distinct complex roots.*

Recall that a version of *Gauß' lemma* states that if $D \in \mathbb{Z}[X]$ divides $F \in \mathbb{Z}[X]$ and the coefficients of $D$ are coprime, then the quotient $F/D$ is a polynomial in $\mathbb{Z}[X]$ as well.

**Lemma 2.** *If two integer polynomials share a non-constant common factor in $\mathbb{C}[X]$, then they also share a non-constant common factor in $\mathbb{Z}[X]$.*

*Proof.* Assume $F, G \in \mathbb{Z}[X]$ have a non-constant common factor in $\mathbb{C}[X]$. This means that their monic greatest common divisor $D \in \mathbb{C}[X]$ is non-constant. Note that, in fact, $D \in \mathbb{Q}[X]$ because it can be computed with the Euclidean algorithm from the coefficients of $F$ and $G$ using only addition, subtraction, multiplication and division. By clearing denominators in $D$ and removing common factors from its coefficients, we can assume that $D$ is an integer polynomial whose coefficients are coprime. By *Gauß' lemma*, $D$ also divides both $F$ and $G$ in $\mathbb{Z}[X]$. □

*Proof (of Proposition 1).* We argue by contradiction. Suppose that the lift $F$ has a multiple complex root $\omega$. Then, $\omega$ is a common root of $F$ and its derivative $F'$, so these two polynomials share a non-trivial common factor in $\mathbb{C}[X]$. By Lemma 2, they therefore share a non-trivial common factor $G \in \mathbb{Z}[X]$. Reducing modulo $p$, we obtain a polynomial $G \bmod p \in \mathbb{F}_p[X]$ that divides both $F \bmod p = f$ and $F' \bmod p = f'$. Since $F$ and $f$ have the same degree, the leading coefficient of $F$ is not divisible by $p$; hence the same holds for the divisor $G$, and in particular $G \bmod p$ is non-constant. Because $f$ is irreducible and divisible by $G \bmod p$, we must have $f = a \cdot (G \bmod p)$ for some $a \in \mathbb{F}_p^\times$; in particular, $G \bmod p$ and $f$ have the same degree. However, this is impossible: $G \bmod p$ also divides $f'$ whose degree is strictly smaller than that of $f$. □

## B Bound on Polynomial Products

We provide a proof for the following:

**Proposition 6.** *Let $F \in \mathbb{C}[X]$ be a monic polynomial of degree $r$. Then, for all $P, Q \in \mathbb{C}[X]/(F)$, we have $\|PQ\|_\infty \leqslant \mathcal{C}_F \cdot \|P\|_\infty \cdot \|Q\|_\infty$, where the constant $\mathcal{C}_F$ is defined as $\mathcal{C}_F = r \cdot (1 + \|F\|_\infty)^{r-1}$.*

*Proof.* Consider the natural projection $\pi : \mathbb{C}[X] \to \mathbb{C}[X]/(F)$ and let $c = \mathcal{C}_F/r$. We first prove that for every polynomial $R \in \mathbb{C}[X]$ of degree at most $2r-2$, we have $\|\pi(R)\|_\infty \leqslant c \cdot \|R\|_\infty$. This is enough: taking $R = \hat{P}\hat{Q}$ for lifts $\hat{P}, \hat{Q} \in \mathbb{C}[X]$ of $P$ and $Q$ of degrees at most $r-1$ gives $\|R\|_\infty \leqslant r\|P\|_\infty\|Q\|_\infty$ (each coefficient of $R$ is a sum of at most $r$ products of one coefficient of $P$ with one coefficient of $Q$), and so $\|PQ\|_\infty = \|\pi(R)\|_\infty \leqslant c \cdot \|R\|_\infty \leqslant \mathcal{C}_F \cdot \|P\|_\infty \cdot \|Q\|_\infty$.

We write $F = X^r + \sum_{u=0}^{r-1} F_u X^u$, so that $\pi(X^r) = -\sum_{u=0}^{r-1} F_u \pi(X^u)$ and thus $\|\pi(X^r)\|_\infty \leqslant \|F\|_\infty$. For every polynomial $S = \sum_{u=0}^{r-1} S_u X^u$, we have $\pi(XS) = \sum_{u=1}^{r-1} S_{u-1} X^u - S_{r-1} \sum_{u=0}^{r-1} F_u X^u$, and its infinity norm satisfies $\|\pi(XS)\|_\infty \leqslant$

$\|S\|_\infty(1+\|F\|_\infty)$. By an inductive argument, it then follows that for every $i \in [\![r-1]\!]$, we have $\left\|\pi\left(X^{r+i}\right)\right\|_\infty \leqslant \|F\|_\infty(1+\|F\|_\infty)^i$.

For $R = \sum_{i=0}^{2r-2} R_i X^i$, we have $\pi(R) = \sum_{i=0}^{r-1} R_i X^i + \sum_{i=0}^{r-2} R_{r+i}\pi\left(X^{r+i}\right)$. Taking infinity norms yields $\|\pi(R)\|_\infty \leqslant \|R\|_\infty + \sum_{i=0}^{r-2} \|R\|_\infty \cdot \left\|\pi\left(X^{r+i}\right)\right\|_\infty$. Applying the bounds on $\left\|\pi\left(X^{r+i}\right)\right\|_\infty$ then gives:

$$\|\pi(R)\|_\infty \leqslant \|R\|_\infty + \|R\|_\infty\|F\|_\infty \sum_{i=0}^{r-2}(1+\|F\|_\infty)^i.$$

With the formula for the sum of a geometric sum, we get $\|\pi(R)\|_\infty \leqslant \|R\|_\infty + \|R\|_\infty\left((1+\|F\|_\infty)^{r-1}-1\right) = c \cdot \|R\|_\infty$. □

## C Details on Error Bounds

In this section, we provide a proof for the following:

**Proposition 9.** *Let $g : \mathbb{C}^m \to \mathbb{C}^n$ be a multivariate polynomial in the real and imaginary parts of the input slots. Let $\boldsymbol{z} \in \mathbb{C}^m$ be a random input vector sampled according to some distribution $\mathcal{D}$ on $\mathbb{C}^m$ with bounded support, and let $\boldsymbol{e} \in \mathbb{C}^m$ be an input error vector independent of $\boldsymbol{z}$ with independent slots following centered complex Gaussian distributions of variance at most $\sigma^2$. Then the slots of the output error vector $g(\boldsymbol{z}+\boldsymbol{e}) - g(\boldsymbol{z})$ have variance $\mathcal{O}\left(\sigma^2\right)$ as $\sigma \to 0$, where the hidden constant only depends on $g$ and $\mathcal{D}$.*

First, we prove a version of the result for real-valued maps, and then we deduce Proposition 9 from it:

**Proposition 11.** *Let $g : \mathbb{R}^m \to \mathbb{R}^n$ be a multivariate polynomial in the input slots. Let $\boldsymbol{x} \in \mathbb{R}^m$ be a random input vector sampled according to some distribution $\mathcal{D}$ on $\mathbb{R}^m$ with bounded support, and let $\boldsymbol{e} \in \mathbb{R}^m$ be an input error vector independent of $\boldsymbol{x}$ with independent slots following centered real Gaussian distributions of variance at most $\sigma^2$. Then, the slots of the output error vector $g(\boldsymbol{x}+\boldsymbol{e}) - g(\boldsymbol{x})$ have variance $\mathcal{O}\left(\sigma^2\right)$ as $\sigma \to 0$, where the hidden constant only depends on $g$ and $\mathcal{D}$.*

*Proof.* Since we can prove the result for each output slot separately, we can assume that $g : \mathbb{R}^m \to \mathbb{R}$. Denote the slots of $\boldsymbol{e}$ by $e_i$ for $i \in [\![m]\!]$ and the partial derivative of $g$ with respect to its $i$-th input slot by $\partial_i g$. Then, the Taylor approximation of $g$ around $\boldsymbol{x}$ yields $g(\boldsymbol{x}+\boldsymbol{e}) - g(\boldsymbol{x}) = L(\boldsymbol{x},\boldsymbol{e}) + R(\boldsymbol{x}+\boldsymbol{e})$, where $L(\boldsymbol{x},\boldsymbol{e}) = \sum_{i=0}^{m-1} \partial_i g(\boldsymbol{x}) \cdot e_i$ is the linear part in $\boldsymbol{e}$ and where the remainder term satisfies $|R(\boldsymbol{x}+\boldsymbol{e})| = \mathcal{O}\left(\|\boldsymbol{e}\|_2^2\right)$ as $\boldsymbol{e} \to \boldsymbol{0}$. We will prove that $\mathbb{E}\left[\left(R(\boldsymbol{x}+\boldsymbol{e})\right)^2\right] = \mathcal{O}\left(\sigma^4\right)$ as $\sigma \to 0$ and that $\mathbb{E}\left[(L(\boldsymbol{x},\boldsymbol{e}))^2\right] = \mathcal{O}\left(\sigma^2\right)$ as $\sigma \to 0$. From there, the result follows: using that $\mathrm{Var}(y) \leqslant \mathbb{E}\left[y^2\right]$ for any real random variable $y$ and that

$(a+b)^2 \leqslant 2a^2 + 2b^2$ for any real $a$ and $b$, we have as $\sigma \to 0$:

$$\begin{aligned}\mathrm{Var}(g(\boldsymbol{x}+\boldsymbol{e}) - g(\boldsymbol{x})) &= \mathrm{Var}(L(\boldsymbol{x},\boldsymbol{e}) + R(\boldsymbol{x}+\boldsymbol{e})) \\ &\leqslant \mathbb{E}\big[(L(\boldsymbol{x},\boldsymbol{e}) + R(\boldsymbol{x}+\boldsymbol{e}))^2\big] \\ &\leqslant 2\mathbb{E}\big[L(\boldsymbol{x},\boldsymbol{e})^2\big] + 2\mathbb{E}\big[R(\boldsymbol{x}+\boldsymbol{e})^2\big] \\ &= \mathcal{O}\big(\sigma^2\big) + \mathcal{O}\big(\sigma^4\big) \\ &= \mathcal{O}\big(\sigma^2\big).\end{aligned}$$

For the first part, note that $(R(\boldsymbol{x}+\boldsymbol{e}))^2 = \mathcal{O}\big(\|\boldsymbol{e}\|_2^4\big)$ as $\boldsymbol{e} \to \boldsymbol{0}$ (importantly, the hidden constant can be chosen uniformly for any $\boldsymbol{x}$ within the bounded support of $\mathcal{D}$), and this implies that $\mathbb{E}\big[(R(\boldsymbol{x}+\boldsymbol{e}))^2\big] = \mathcal{O}\big(\mathbb{E}\big[\|\boldsymbol{e}\|_2^4\big]\big) = \mathcal{O}\big(\sigma^4\big)$ as $\sigma \to 0$. For the second part, we compute:

$$\mathbb{E}\big[(L(\boldsymbol{x},\boldsymbol{e}))^2\big] = \mathbb{E}\left[\left(\sum_{i=0}^{m-1} \partial_i g(\boldsymbol{x}) \cdot e_i\right)^2\right] = \sum_{i,j=0}^{m-1} \mathbb{E}[\partial_i g(\boldsymbol{x}) \partial_j g(\boldsymbol{x}) \cdot e_i e_j].$$

Using independence, we get:

$$\mathbb{E}\big[(L(\boldsymbol{x},\boldsymbol{e}))^2\big] = \sum_{i=0}^{m-1} \mathbb{E}\big[(\partial_i g(\boldsymbol{x}))^2\big] \cdot \mathbb{E}\big[e_i^2\big] \leqslant \sigma^2 \cdot \sum_{i=0}^{m-1} \mathbb{E}\big[(\partial_i g(\boldsymbol{x}))^2\big] = \mathcal{O}\big(\sigma^2\big). \qquad \square$$

*Proof (of Proposition 9).* We identify $\mathbb{C}^m$ and $\mathbb{C}^n$ with $\mathbb{R}^{2m}$ and $\mathbb{R}^{2n}$, respectively. Under this identification, the map $g$ can be viewed as a real polynomial map $g : \mathbb{R}^{2m} \to \mathbb{R}^{2n}$. The complex input error vector $\boldsymbol{e}$ then corresponds to a real vector with independent, centered Gaussian slots, each of variance at most $\sigma^2/2 \leqslant \sigma^2$. Applying Proposition 11 to this real representation shows that each slot of the output error $g(\boldsymbol{z}+\boldsymbol{e}) - g(\boldsymbol{z})$ has variance $\mathcal{O}\big(\sigma^2\big)$ when viewed as a vector in $\mathbb{R}^{2n}$. Grouping pairs of real slots back into complex slots, we conclude that each complex slot of the output error vector has variance of twice that amount, which is still $\mathcal{O}\big(\sigma^2\big)$. $\square$

## D Incorporating Matrix into Slot-To-Coeff Transformation

We show here how to incorporate a matrix multiplication into the first layer of the S2C procedure of [13], thereby saving one multiplicative level.

**Lemma 3.** *Let $\boldsymbol{M}, \boldsymbol{N} \in \mathbb{C}^{n \times n}$ be two matrices such that* $\mathsf{DiagSupp}(\boldsymbol{M})$ *and* $\mathsf{DiagSupp}(\boldsymbol{N})$ *are contained in intervals modulo $n$ of lengths $D_{\boldsymbol{M}}, D_{\boldsymbol{N}} \geqslant 1$, respectively. Then,* $\mathsf{DiagSupp}(\boldsymbol{MN})$ *is contained in an interval modulo $n$ of length at most $D_{\boldsymbol{M}} + D_{\boldsymbol{N}} - 1$.*

*Proof.* Write $\boldsymbol{M} = (M_{i,j})_{i,j \in [\![n]\!]}$ and $\boldsymbol{N} = (N_{i,j})_{i,j \in [\![n]\!]}$. Let $I_{\boldsymbol{M}}$ and $I_{\boldsymbol{N}}$ be intervals modulo $n$ of lengths $D_{\boldsymbol{M}}$ and $D_{\boldsymbol{N}}$ which contain $\mathsf{DiagSupp}(\boldsymbol{M})$ and $\mathsf{DiagSupp}(\boldsymbol{N})$, respectively. Consider a nonzero entry of $\boldsymbol{MN}$, say in position

$(i, j)$. Then there exists $k \in [\![n]\!]$ such that $M_{i,k} \neq 0$ and $N_{k,j} \neq 0$. Hence, $(k - i) \bmod n \in I_{\boldsymbol{M}}$ and $(j - k) \bmod n \in I_{\boldsymbol{N}}$. Since $j - i = (k - i) + (j - k)$, we obtain $(j - i) \bmod n \in (I_{\boldsymbol{M}} + I_{\boldsymbol{N}}) \bmod n$. Therefore, $\mathsf{DiagSupp}(\boldsymbol{MN})$ is contained in $(I_{\boldsymbol{M}} + I_{\boldsymbol{N}}) \bmod n$, which is an interval modulo $n$ of length at most $D_{\boldsymbol{M}} + D_{\boldsymbol{N}} - 1$. □

The authors of [13] propose a variant of the S2C procedure that decomposes it into several layers via a Cooley–Tukey factorization [19]. More precisely, the complex matrix underlying S2C is decomposed into $\rho \in [\![1, \log_2 n]\!]$ matrices $\boldsymbol{E}_0, \ldots, \boldsymbol{E}_{\rho-1} \in \mathbb{C}^{n \times n}$. When applied homomorphically, the procedure consumes $\rho$ multiplicative levels and performs $\mathcal{O}\big(\rho \cdot n^{1/\rho}\big)$ ring operations in $\mathcal{R}$.

The first layer $\boldsymbol{E}_0$ has diagonal support contained in an interval modulo $n$ of length $\mathcal{O}\big(n^{1/\rho}\big)$. Now let $\boldsymbol{M} \in \mathbb{C}^{n \times n}$ be a matrix whose diagonal support is contained in an interval modulo $n$ of length $D$. By Lemma 3, the product $\boldsymbol{E}_0\boldsymbol{M}$ has diagonal support contained in an interval modulo $n$ of length $\mathcal{O}\big(n^{1/\rho} + D\big)$. Using the baby-step giant-step method, we can therefore homomorphically apply $\boldsymbol{E}_0\boldsymbol{M}$ using $\mathcal{O}\big(n^{1/\rho} + D\big)$ plaintext–ciphertext multiplications and $\mathcal{O}\big(\sqrt{n^{1/\rho} + D}\big)$ rotations. This matches the asymptotic complexity of first homomorphically multiplying by $\boldsymbol{M}$ and then applying the first S2C layer separately, while saving one multiplicative level.