Efficient Large-Integer Arithmetic for FHE

Ahmad Al Badawi[1] Andreea Alexandru[1] Gurgen Arakelov[2]
Charles Gouert[3] Sergey Gomenyuk[2] Valentina Kononova[2] Yarkın Doröz[3]
Yuriy Polyakov[1]

[1]Duality Technologies [2]Fair Math [3]NVIDIA

Abstract

Fully Homomorphic Encryption (FHE) has emerged as one of the key technologies for privacy-preserving computation, enabling arbitrary computation directly on encrypted data. Vectorized FHE schemes, such as Brakerski/Fan–Vercauteren (BFV), Brakerski–Gentry–Vaikuntanathan (BGV), and Cheon–Kim–Kim–Song (CKKS), are typically used in applications dealing with large datasets, for example, confidential database queries and private ML inference. These FHE schemes are based on the computational hardness of Ring Learning with Errors (RLWE) and share a common algebraic foundation: arithmetic over high-dimensional polynomial rings with coefficient moduli spanning hundreds or thousands of bits, far exceeding the native arithmetic capabilities of modern processors.

This article surveys the evolution of large-integer arithmetic in RLWE-based FHE libraries, with a focus on the Residue Number System (RNS) techniques used in practically all modern implementations. We give a formal treatment of the two fundamental RNS building blocks — *basis extension* and *scaling* — that require information about the *magnitude* of a large value and are therefore incompatible with a purely residue-wise view of arithmetic. We contrast the two principal algorithmic approaches to these operations: the integer-only approach of Bajard, Eynard, Hasan, and Zucca (BEHZ), which tolerates approximation overflows and corrects them with auxiliary redundant moduli, and the floating-point approach of Halevi, Polyakov, and Shoup (HPS). We then show how these primitives compose into the higher-level RNS procedures used across all vectorized RLWE schemes and review how their adoption reshaped the architecture and performance of libraries such as HElib, SEAL, PALISADE/OpenFHE, HEAAN, and Lattigo. We give particular attention to the scaling error inherent in the original Full RNS variant of CKKS, and to the more recent techniques — reduced-error scaling, composite scaling, and grafting — that eliminate it or restore flexible, high-precision rescaling from within the residue representation. We also cover GPU-accelerated implementations and close by discussing a renewed, and so far exploratory, interest in positional (non-RNS) representations, raising the question of how such approaches might compare with the Full RNS variants that dominate FHE implementations today.

Contents

1 Introduction **3**

2 Large-Integer Arithmetic in RLWE-based Schemes **4**
2.1 Polynomial rings and notation 4
2.2 Practical parameters 4

3 The First Generation: Arbitrary-Precision Arithmetic **5**

4 From Partial to Full RNS: the Library Evolution **6**
4.1 Remaining problems of the partial-RNS stage 7
4.2 Full RNS 7

5 RNS Toolkit: Representation, Basis Extension, and Scaling **8**
5.1 CRT representation 8
5.2 Why magnitude is the fundamental obstacle 9
5.3 Fast basis extension 11
5.3.1 The BEHZ approach: approximate conversion with integer-only correction 11
5.3.2 The HPS approach: exact conversion via floating-point estimation 12
5.4 Simple scaling in RNS 14
5.4.1 HPS simple scaling 14
5.4.2 BEHZ simple scaling 14
5.5 Complex scaling: scaling out of an extended base 15
5.6 BEHZ versus HPS: a summary 15

6 Composed RNS Procedures Across Schemes **16**
6.1 Modulus switching and CKKS rescaling 16
6.2 Key switching 18
6.2.1 Hybrid key switching in RNS 19
6.3 BFV-specific composition 20

7 GPU Acceleration of RLWE-based FHE **20**

8 A Renewed Interest in Positional Arithmetic **22**

9 Conclusion **23**

1 Introduction

The rapid growth of AI has significantly increased the demand for technologies capable of processing sensitive data while preserving user privacy. In many practical scenarios, including healthcare, finance, cloud computing, and edge AI, machine learning models must operate on confidential information that cannot be exposed to service providers or third parties. Fully Homomorphic Encryption (FHE) addresses this challenge by enabling arbitrary computations to be performed directly on encrypted data, allowing such workloads to be executed without revealing the underlying information [Gen09].

Most modern FHE libraries are built upon the family of vectorized Ring Learning with Errors (RLWE)-based encryption schemes, including the Brakerski/Fan–Vercauteren scheme (BFV) [Bra12, FV12], the Brakerski–Gentry–Vaikuntanathan scheme (BGV) [BGV12], and the Cheon–Kim–Kim–Song scheme (CKKS) [CKKS17]. Although these schemes target different computational models — exact integer arithmetic in the case of BFV and BGV, and approximate fixed-point arithmetic in the case of CKKS — they share a common algebraic foundation and many implementation techniques. In particular, they all rely on arithmetic over high-dimensional polynomial rings with large coefficient moduli, making efficient modular arithmetic one of the fundamental building blocks of every RLWE-based FHE library.

While the underlying cryptographic algorithms are mathematically well defined, their practical implementation requires numerous engineering decisions that directly affect performance, memory consumption, and software complexity. Among these decisions, the representation and manipulation of large coefficient moduli play a central role. At practical security levels, vectorized RLWE-based schemes operate on polynomial rings whose coefficient moduli span hundreds or even thousands of bits, whereas contemporary processors efficiently support arithmetic only on fixed-width integers, typically up to 64 bits. Efficient software implementations must therefore bridge a substantial gap between the mathematical model of the scheme and the capabilities of modern hardware.

Over the past decade, the implementation of large-integer arithmetic in FHE libraries has undergone a significant evolution. Early implementations relied primarily on arbitrary-precision integer arithmetic, while later generations increasingly adopted Residue Number System (RNS) representations and specialized modular arithmetic algorithms. The turning point in this evolution was the realization that the only operations that genuinely resist a residue-wise implementation are those that depend on the *magnitude* of the represented value — comparisons against the dynamic range, division, and rounding. All such operations reduce to two primitives: *basis extension* and *scaling* in the Chinese Remainder Theorem (CRT) representation. Two competing algorithmic families for these primitives emerged around 2016–2018: the integer-only techniques of Bajard, Eynard, Hasan, and Zucca (BEHZ) [BEHZ16] and the floating-point techniques of Halevi, Polyakov, and Shoup (HPS) [HPS19], whose practical trade-offs were analyzed in detail in [BPA+21].

This article surveys this evolution. Rather than focusing on the cryptographic construction of individual schemes, we examine how the implementation strategies evolved, the engineering trade-offs that motivated their adoption, and the advantages and limitations of each approach. In contrast to prior expositions that are tied to a particular scheme, we deliberately keep the presentation scheme-generic: the primitives of Section 5 apply equally to BFV, BGV, and CKKS, and Section 6 shows how they compose into the higher-level procedures shared by all vectorized RLWE schemes. CKKS is used as the primary motivating example where a concrete scheme is helpful, due to its widespread adoption in privacy-preserving machine learning.

The remainder of the article is organized as follows. Section 2 introduces the computational model of RLWE-based schemes and quantifies the large-integer arithmetic problem. Section 3 reviews the first generation of libraries based on multiprecision arithmetic. Section 4 traces the historical transition from partial to Full RNS architectures across major libraries. Sections 5 and 6 then present the technical substance behind this evolution: Section 5 formalizes the RNS toolkit — representation, basis extension, and scaling, contrasting the BEHZ and HPS approaches — and Section 6 presents the composed procedures — modulus switching/CKKS rescaling and hybrid key switching — built from it. Section 7 outlines GPU acceleration, Section 8 discusses a re-emerging interest in alternative representations to Full RNS, and Section 9 provides concluding remarks.

2 Large-Integer Arithmetic in RLWE-based Schemes

2.1 Polynomial rings and notation

All RLWE-based encryption schemes considered here operate over the cyclotomic polynomial ring

$$\mathcal{R}_Q \;=\; \mathbb{Z}_Q[X]/(X^N+1), \tag{1}$$

where the ring dimension N is a power of two and Q is the coefficient modulus. Ciphertexts are tuples of polynomials whose coefficients belong to $\mathbb{Z}_Q$. Every homomorphic operation — addition, multiplication, modulus switching/rescaling, key switching, and bootstrapping — ultimately reduces to arithmetic on these coefficients. Consequently, the computational cost of an FHE workload is dominated by repeated modular additions, multiplications, and reductions modulo Q.

Throughout the paper, we identify $\mathbb{Z}_n$ with its representation in the symmetric interval $\mathbb{Z} \cap [-n/2, n/2)$, and denote by $[x]_n$ the centered reduction of x into that interval. We write $|x|_n \in \{0, \ldots, n-1\}$ for the standard (non-negative) representative when the distinction matters. We denote by $\lfloor x \rfloor$, $\lceil x \rceil$, and $\lceil x \rfloor$ rounding down, up, and to the nearest integer, respectively, and extend these notations to vectors and polynomial coefficients element-wise.

Unlike conventional cryptographic algorithms, where the modulus is chosen to fit within a machine word, RLWE-based schemes require coefficient moduli that are substantially larger than the native integer types supported by modern processors. The need to efficiently manipulate these large integers lies at the core of every practical FHE implementation. Modern implementations bridge this gap by choosing the coefficient modulus as a product of pairwise coprime machine-word-sized primes, $Q = \prod_i q_i$ — a decomposition we use informally from Section 4 onward and treat formally in Section 5, where a generic modulus of this form is written in lowercase as $q = \prod_{i=1}^{k} q_i$.

2.2 Practical parameters

The size of the coefficient modulus depends on the desired security level, the multiplicative depth of the evaluated circuit, and the precision requirements of the target application. For a given ring dimension N and security level, lattice security analysis imposes an upper bound on the total ciphertext modulus; applications then choose a modulus below this bound according to their depth and precision needs. Table 1 lists the maximal modulus sizes at the 128-bit security level for uniform ternary secrets, following Table 5.2 of the homomorphic encryption security guidelines [BCC+24], together with the functionality typically instantiated at each ring dimension.

Table 1: Maximal ciphertext modulus size $\log_2 Q$ per ring dimension N at the 128-bit security level, for uniform ternary secret and Gaussian error with $\sigma = 3.19$ (from Table 5.2 of [BCC+24]), with the functionality typically instantiated at each dimension.

Ring dimension N	Max. $\log_2 Q$	Typical functionality	Representative applications
4096 8192	106 214	Shallow evaluation	PSI (e.g., Microsoft APSI), PIR, secure database queries, secure analytics
16384	430	Intermediate-depth evaluation	Complex pipelines, ML inference
32768 65536 131072	868 1747 3523	Bootstrapping, deep computations	CNN inference (ResNet/VGG), logistic regression training, private LLM inference

These bounds are consistent with the default parameter validation performed in modern RLWE-based FHE libraries such as OpenFHE [AAB+22]. They illustrate an important observation: practical FHE implementations routinely perform arithmetic over integers consisting of several hundred or even several thousand bits. The scale of the computation becomes even more apparent when considering the size of a ciphertext. For instance, a ciphertext with $N = 2^{14}$ consists of two or more polynomials, each containing 16,384 coefficients modulo an integer of several hundred bits; a single homomorphic multiplication therefore requires millions of modular operations over integers far larger than the 64-bit words natively supported by commodity processors.

This mismatch between the mathematical requirements of RLWE-based cryptography and the capabilities of contemporary hardware has been one of the primary driving forces behind the evolution of FHE implementations, from arbitrary-precision arithmetic to architectures based on the Residue Number System.

3 The First Generation: Arbitrary-Precision Arithmetic

The earliest implementations of RLWE-based FHE libraries relied on arbitrary-precision (multiprecision) integer arithmetic to represent coefficients modulo Q larger than the native machine word size. This approach closely followed the mathematical formulation of the schemes and allowed implementations to reuse mature multiprecision libraries instead of developing specialized finite-field arithmetic from scratch. This generation occupies the leftmost band of Figure 1, which chronologically illustrates the main algorithmic milestones of multiprecision and RNS arithmetic.

One of the first practical FHE libraries, HElib [HS20], used NTL [Sho] for polynomial and large-integer arithmetic, with NTL itself relying on GMP [Gt] as its arbitrary-precision backend. Even at this stage, HElib's internal representation was a hybrid one — its *DoubleCRT* representation, discussed in Section 4, already employed RNS techniques for selected operations while still depending on multiprecision arithmetic elsewhere, foreshadowing the transition toward the fully RNS-based architectures adopted by later libraries. Other early libraries, including the initial versions of Microsoft SEAL [Mic] and PALISADE [PRR], also targeted BFV using multiprecision arithmetic before evolving toward RNS-native implementations. The growing computational demands of homomorphic encryption, together with initiatives such as the DARPA PROCEED program [DAR], motivated substantial research into more efficient multiprecision

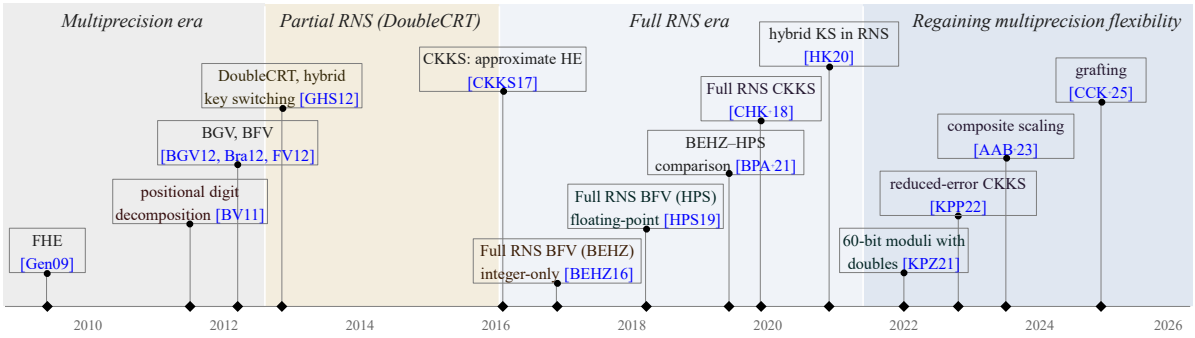


Figure 1: Algorithmic milestones in large-integer arithmetic for FHE. Each diamond marks one contribution, placed at the approximate date of its *first public appearance* — the ePrint or arXiv posting where one exists, otherwise the conference at which it was presented. Background bands mark the architectural generations: the multiprecision era (Section 3), the partial-RNS stage and the transition to Full RNS (Section 4, with the primitives formalized in Section 5), and the later milestones that recover, from within the residue representation, the flexibility of the multiprecision world — exact scales, large scaling factors, and arbitrary rescaling (Section 6.1).

arithmetic and RNS-based techniques.

The advantages of the multiprecision approach were a natural correspondence with the mathematical formulation, straightforward implementation, easy correctness verification, and reuse of mature, well-tested arithmetic libraries. Its limitations, however, were significant: low performance, frequent cache misses, expensive modular multiplication and reduction, and dynamic memory management overhead. Although this architecture provided a clean and mathematically intuitive implementation, the overhead of arbitrary-precision arithmetic quickly became the primary performance bottleneck for practical FHE workloads, motivating the transition toward RNS-based implementations described next.

4 From Partial to Full RNS: the Library Evolution

Early RLWE-based FHE libraries, including HElib, Microsoft SEAL, and PALISADE, already employed the polynomial CRT decomposition, implemented via the Number Theoretic Transform (NTT), for efficient polynomial arithmetic. However, arithmetic over the large coefficient modulus continued to rely on arbitrary-precision integer libraries, making multiprecision arithmetic one of the primary computational bottlenecks. Applying the CRT to the coefficient modulus itself — a representation known as the integer CRT, or RNS — replaced expensive multiprecision operations with collections of independent modular operations over small integers. The transition traced in this section corresponds to the second and third bands of Figure 1.

HElib was the first FHE library to adopt this idea through its *DoubleCRT* representation [HS20, GHS12]. Instead of storing a polynomial by its coefficients modulo the large integer Q, DoubleCRT represents the polynomial separately modulo each small prime q_i, with each residue stored in NTT form. The name reflects the use of two independent CRT decompositions: one over the polynomial ring and another over the coefficient modulus. This hybrid representation significantly accelerated polynomial multiplication and other component-wise operations while

substantially reducing the reliance on arbitrary-precision arithmetic.

4.1 Remaining problems of the partial-RNS stage

Although integer CRT dramatically reduced the cost of modular arithmetic, it did not eliminate all sources of overhead. The choice of residue moduli became constrained: each q_i typically had to be NTT-friendly, providing the necessary roots of unity for the selected polynomial degree, so parameter selection became considerably more restrictive than in the arbitrary-precision setting. Moreover, while polynomial multiplication could be performed efficiently in the CRT/NTT representation, many other operations still required conversions between representations: libraries frequently performed inverse NTTs to recover coefficient representations before executing certain algorithms, followed by forward NTTs to continue in the transform domain. Most importantly, the operations that depend on the *magnitude* of a coefficient as an integer — the divide-and-round and base-decomposition steps inside decryption, multiplication, modulus switching, and key switching (formalized in Section 5.2) — still reverted to positional multiprecision arithmetic. The implementation was therefore hybrid: some parts of the computation lived naturally in the residue domain, while others depended on global magnitude information about the large modulus. A substantial fraction of the computation was spent on representation conversions rather than on the arithmetic itself.

4.2 Full RNS

A major step toward removing this hybrid structure was the BEHZ Full RNS variant of the BFV scheme [BEHZ16], which showed how to replace the remaining multiprecision operations with residue-wise algorithms for basis extension and scaling, performing the central operations of BFV-like schemes directly in RNS form (the algorithms themselves are presented in detail in Sections 5.3.1 and 5.5). The BEHZ techniques were subsequently adopted by Microsoft SEAL, which became the first major FHE library to implement a Full RNS variant of BFV (versions $\geq$ 2.3.0). This direction was refined by the HPS variant [HPS19], implemented in PALISADE (v1.1+), which simplified the formulation and reduced the cost of basis extension and scaling; the implementation study of [BPA+21] compared both variants experimentally on CPUs and GPUs, initially observing better noise scaling for HPS with increasing multiplicative depth — a gap later traced by Bajard et al. [BEM+19] to an implementation deviation in the BEHZ correction procedure rather than to the variant itself (discussed further in Section 5.6).

For approximate homomorphic encryption the transition was more gradual. The first implementation of CKKS in the HEAAN library relied on arbitrary-precision arithmetic, using DoubleCRT primarily for polynomial multiplication while performing several higher-level operations on large integers. Since CKKS relies on rescaling after every multiplication — an operation central to both correctness and numerical precision — extending Full RNS to approximate arithmetic required new techniques for performing rescaling and modulus switching directly in the RNS domain while preserving accuracy. Cheon, Han, Kim, Kim, and Song addressed this with a Full RNS variant of CKKS featuring an RNS-friendly modulus structure and an approximate rescaling procedure performed entirely on residues (presented in Section 6.1) [CHK+18]. These ideas became the foundation of modern high-performance CKKS implementations; around the same period, several research groups independently developed Full RNS CKKS implementations in SEAL, PALISADE [BGP+20], and HElib [BHHH19], many of which were evaluated in the iDASH 2018 homomorphic encryption competition [KJT+20].

As a result, Full RNS became the dominant backend architecture for modern RLWE-based FHE libraries. Libraries such as Microsoft SEAL, OpenFHE, and Lattigo [Tun23] rely on RNS decompositions, NTT-friendly moduli, and residue-wise modular arithmetic as the default implementation model for BFV, BGV, and CKKS. The main achievement of this evolution is that ciphertexts remain in RNS form across (nearly) all operations, avoiding repeated reconstruction into large integers and enabling efficient use of native machine-word arithmetic and SIMD instructions.

At the same time, Full RNS introduced its own limitations and inherited one. The NTT-friendliness constraint of the partial-RNS stage (Section 4.1) remains: the modulus chain must be built from NTT-friendly primes, restricting parameter selection and coupling the arithmetic backend to the ring dimension. Basis extension, modulus switching, key switching, and bootstrapping require substantial precomputation and careful memory management. For CKKS, rescaling granularity is determined by the bit sizes of the available RNS primes (see Section 6.1), making precision management and parameter selection more rigid, and the mismatch between NTT primes and the target scaling factor introduces an inherent scaling error; further improvements such as reduced-error scale management [KPP22], composite scaling [AAB+23] (OpenFHE), and grafting [CCK+25] (HEAAN) eliminate the error and restore flexibility at the cost of additional complexity, as discussed in Section 6.1. Thus, Full RNS solved the primary performance bottlenecks of arbitrary-precision arithmetic but shifted much of the implementation complexity to modulus-chain construction, basis management, precision control, and memory-efficient data movement. Modern FHE libraries are not merely "using RNS"; their performance depends critically on how efficiently the entire RNS backend is engineered. The remainder of this article develops that backend in technical detail: the next section formalizes the two primitives at its core — basis extension and scaling — and contrasts the BEHZ and HPS realizations, and Section 6 shows how the higher-level procedures named throughout this evolution (modulus switching, rescaling, hybrid key switching) are assembled from them.

5 RNS Toolkit: Representation, Basis Extension, and Scaling

The historical account of Section 4 identified RNS basis extension and scaling as the algorithmic advances that made Full RNS architectures possible. This section presents these primitives formally: we introduce the CRT representation and its reconstruction identities, explain why magnitude-dependent operations are the fundamental obstacle to a purely residue-wise implementation, and then present the two competing realizations of the primitives — the integer-only BEHZ approach and the floating-point HPS approach — concluding with a summary of their trade-offs.

5.1 CRT representation

The Residue Number System is a non-positional number system in which an integer is represented by a tuple of residues modulo a set of predefined pairwise coprime moduli, known as the *RNS base*. The motivation is straightforward: modern processors are highly optimized for fixed-width integer arithmetic, typically up to 64 bits. While FHE coefficient moduli span hundreds or thousands of bits, they can be chosen as the product of several smaller pairwise coprime moduli, allowing arithmetic to be performed using native machine instructions.

Throughout this section we fix an RNS base $\mathcal{B} = \{q_1, \ldots, q_k\}$ of pairwise coprime single-precision moduli, i.e., $\gcd(q_i, q_j) = 1$ for all $i \neq j$, and let $q = \prod_{i=1}^{k} q_i$. By the CRT, the map

$x \mapsto (x_1, \ldots, x_k)$ with $x_i = [x]_{q_i}$ is a ring isomorphism $\mathbb{Z}_q \cong \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_k}$. We write $x \sim (x_1, \ldots, x_k)$ for this *CRT (or RNS) representation*. Additions, subtractions, and multiplications in $\mathbb{Z}_q$ can be performed *component-wise* on the residues, independently and in parallel, entirely in single-precision arithmetic. The quantity q is called the *dynamic range* of the base: as long as $x \in \mathbb{Z} \cap [-q/2, q/2)$, the representation is unique.

The inverse map is given by the CRT reconstruction formula

$$x \;=\; \left[\sum_{i=1}^{k} x_i \cdot \left[\left(\tfrac{q}{q_i}\right)^{-1} \right]_{q_i} \cdot \frac{q}{q_i} \right]_q . \tag{2}$$

Two ways of "opening up" this formula, i.e., making the reduction modulo q explicit, are used pervasively in what follows:

$$x = \left(\sum_{i=1}^{k} \Big[x_i \cdot \big[\left(\tfrac{q}{q_i}\right)^{-1} \big]_{q_i} \Big]_{q_i} \cdot \frac{q}{q_i} \right) - v \cdot q \qquad \text{for some } v \in \mathbb{Z},\ |v| \le k/2, \tag{3}$$

$$x = \left(\sum_{i=1}^{k} x_i \cdot \left[\left(\tfrac{q}{q_i}\right)^{-1} \right]_{q_i} \cdot \frac{q}{q_i} \right) - v' \cdot q \qquad \text{for some } v' \in \mathbb{Z},\ |v'| \le \tfrac{k}{2} \max_i(q_i). \tag{4}$$

In both cases, the correction term (v or v') accounts for the missing final reduction modulo q. These identities are the source of both the power and the difficulty of RNS: the sums can be computed residue-wise, but the integers v, v' encode global *magnitude* information about x that no single residue channel can see.

5.2 Why magnitude is the fundamental obstacle

Component-wise operations — the additions and multiplications that dominate polynomial arithmetic — map perfectly onto RNS. In combination with the NTT applied independently modulo each q_i (recall that each q_i is required to be NTT-friendly, i.e., $q_i \equiv 1 \pmod{2N}$), polynomial multiplication becomes an embarrassingly parallel collection of single-precision kernels.

However, every RLWE-based scheme also contains operations that are *not* ring operations in $\mathbb{Z}_q$. Three recurring patterns are:

(i) **Lifting.** Interpreting $x \in \mathbb{Z}_q$ as an integer in $[-q/2, q/2)$ and re-representing it modulo a larger modulus $Q = q \cdot p$ (e.g., before a BFV tensor product whose result must not wrap around, or when raising a ciphertext digit to an extended basis during key switching).

(ii) **Scaling and rounding.** Computing $y = \left\lfloor \frac{t}{q}\, x \right\rceil$ for some $t \ll q$ (e.g., BFV decryption and multiplication, CKKS rescaling, modulus switching in all schemes, and the final scale-down of key switching).

(iii) **Digit decomposition.** Splitting x into low-norm digits whose recombination is linear (used by all key-switching methods to control noise growth).

All three depend on the magnitude of x as an integer, not merely on its residues, and therefore seem to require conversion back to a positional (multiprecision) representation — which would negate the benefit of RNS altogether. The central contribution of the RNS literature for FHE is a pair of primitives that perform (i) and (ii) *directly on residues*: RNS *basis extension* and

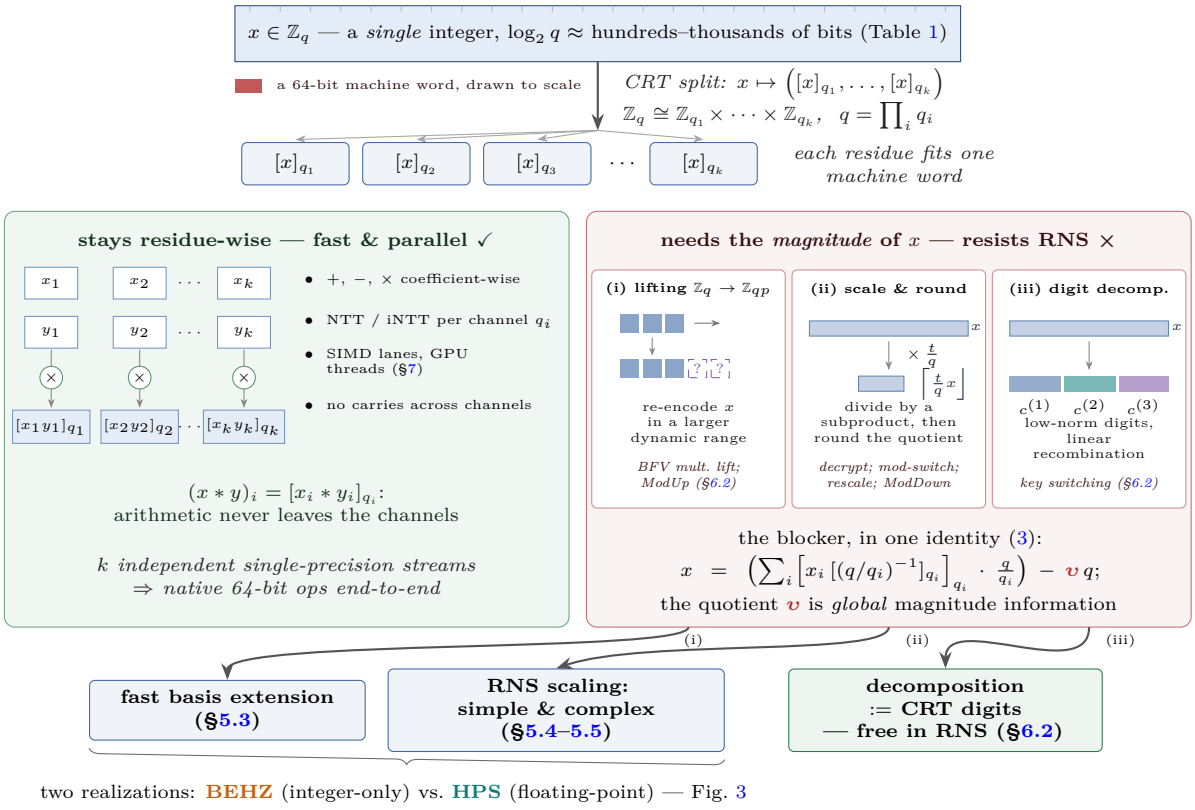


Figure 2: Magnitude as the fundamental obstacle to a purely residue-wise implementation (Section 5.2). A coefficient $x \in \mathbb{Z}_q$, far exceeding the machine word, is split by the CRT into k single-word residue channels. Ring operations (left) proceed channel-by-channel and never leave the residue domain. The three recurring patterns of RLWE-based schemes (right) — lifting, scaling with rounding, and digit decomposition — instead depend on the magnitude of x as an integer: in the reconstruction identity (3) that magnitude is carried by the quotient v, which no single channel can see. The RNS toolkit resolves the conflict with two residue-only primitives (bottom); digit decomposition is sidestepped altogether by taking the CRT residues themselves as the digits.

scaling. Pattern (iii), as we discuss in Section 6.2, was elegantly sidestepped by redefining the decomposition itself to be the RNS decomposition. Figure 2 summarizes this dichotomy and provides pointers to sections describing the respective solutions.

Two algorithmic families realize these primitives. The BEHZ family [BEHZ16] stays entirely within integer arithmetic, accepts that fast conversions are only approximate (they carry a small multiple of q as overflow), and removes the overflow using auxiliary redundant moduli. The HPS family [HPS19] computes the correction quotient v directly using floating-point arithmetic, obtaining (nearly) exact conversions without auxiliary moduli. Figure 3 describes the two dataflows side by side, together with the trade-offs examined in Section 5.6.

5.3 Fast basis extension

Let $x \in \mathbb{Z}_q$ be given in CRT representation $(x_1, \ldots, x_k)$ with respect to $\mathcal{B} = \{q_1, \ldots, q_k\}$, and let $\mathcal{B}' = \{p_1, \ldots, p_{k'}\}$ be a second base, coprime to the first, with $p = \prod_j p_j$. *Basis extension* computes the residues of x (or of a value congruent to x modulo q) with respect to $\mathcal{B}'$, yielding a representation in the extended base $\mathcal{B} \cup \mathcal{B}'$ with dynamic range qp.

5.3.1 The BEHZ approach: approximate conversion with integer-only correction

The BEHZ fast base conversion applies Eq. (3) *without* computing υ:

$$\mathsf{FastBaseConv}(x, \mathcal{B}, \mathcal{B}') \;=\; \left(\left[\sum_{i=1}^{k} \Big[x_i \cdot \big[\big(\tfrac{q}{q_i}\big)^{-1} \big]_{q_i} \Big]_{q_i} \cdot \frac{q}{q_i} \right]_{p_j} \right)_{p_j \in \mathcal{B}'}. \tag{5}$$

Given precomputed constants $\big[q/q_i\big]_{p_j}$, this requires only single-precision operations. It can be thought of as computing the CRT reconstruction without the final reduction modulo the dynamic range: the converted value equals $x + \upsilon q$ for some small υ (the *q-overflow*), i.e., the result represents x only up to a small multiple of q.

For some uses the overflow merely inflates noise and can be tolerated; for others it must be removed. To this end, BEHZ augments the base with two auxiliary redundant moduli: a modulus $\tilde{m}$ used for an input-side correction, and a modulus m_{sk} carried through the entire computation for an output-side correction. The two mechanisms are:

- **Small Montgomery reduction (input-side correction).** Inputs are pre-multiplied by $\tilde{m}$; after conversion, the residue channel modulo $\tilde{m}$ reveals the overflow υ, which is subtracted, and the factor $\tilde{m}$ is divided out (Algorithm 1). The correction leaves a small additive term that must be tracked in the noise analysis.
- **Shenoy–Kumaresan extension (output-side correction).** The residue modulo m_{sk} is used at the end of the computation to recover υ exactly and produce an exact conversion back to the original base [SK89].

We state the small Montgomery reduction in full because its exact specification turns out to be load-bearing: a slight implementation deviation from it is the crux of the noise-growth episode discussed in Section 5.6. The Shenoy–Kumaresan extension, by contrast, is classical and is used as a black box. Note that m_{sk} appears in Algorithm 1 only as a carried residue channel; it plays no role in the reduction itself.

Algorithm 1 $\mathsf{SmMRq}_{\tilde{m}}$: small Montgomery reduction modulo q (BEHZ [BEHZ16])

Require: $c'' = [\tilde{m}\, c]_q + \upsilon\, q$ represented in $\mathcal{B}' \cup \{m_{sk}, \tilde{m}\}$
Ensure: c' in $\mathcal{B}' \cup \{m_{sk}\}$ with $c' \equiv c''\, \tilde{m}^{-1} \pmod{q}$ and $\|c'\|_\infty \le \frac{\|c''\|_\infty}{\tilde{m}} + \frac{q}{2}$

1: $r_{\tilde{m}} \leftarrow \big[-c''_{\tilde{m}} \cdot q^{-1}\big]_{\tilde{m}}$
2: **for** $j \in \mathcal{B}' \cup \{m_{sk}\}$ **do**
3: $\quad c'_j \leftarrow \big|(c''_j + q\, r_{\tilde{m}})\, \tilde{m}^{-1}\big|_j$
4: **end for**
5: **return** c'

The net effect is a fully integer pipeline, at the cost of (a) additional residue channels ($\tilde{m}$, m_{sk}, and, for decryption, a further modulus γ), (b) extra modular multiplications for the correction

steps, and (c) additional terms in the noise analysis contributed by the correction procedures — terms whose practical impact turns out to be small when the procedures are implemented exactly as specified, a point we return to in Section 5.6.

5.3.2 The HPS approach: exact conversion via floating-point estimation

The HPS extension computes the correction quotient v of Eq. (3) directly. Observing that

$$v \;=\; \left\lceil \sum_{i=1}^{k} \frac{1}{q_i} \Big[x_i \cdot \big[\big(\tfrac{q}{q_i}\big)^{-1} \big]_{q_i} \Big]_{q_i} \right\rfloor, \tag{6}$$

one computes $y_i := \big[x_i \cdot \big[(q/q_i)^{-1}\big]_{q_i} \big]_{q_i}$ in single-precision integer arithmetic, the ratios $z_i := y_i/q_i \in [-\frac{1}{2}, \frac{1}{2})$ in floating-point arithmetic, sums them, and rounds. With v in hand, the exact extension follows from Eq. (3) evaluated modulo each target modulus:

$$\mathsf{FastBaseExt}(x, \mathcal{B}, \mathcal{B}') \;=\; \left(\left[\sum_{i=1}^{k} y_i \cdot \Big[\frac{q}{q_i} \Big]_{p_j} \;-\; v \cdot [q]_{p_j} \right]_{p_j} \right)_{p_j \in \mathcal{B}'}, \tag{7}$$

where $\big[q/q_i\big]_{p_j}$ and $[q]_{p_j}$ are precomputed. No auxiliary moduli and no correction pass are needed.

Correctness and precision. The only source of error is the floating-point estimate of v: instead of the exact z_i we compute approximations with per-term error ϵ_i, so the computed $v^* = \lceil \sum_i (z_i + \epsilon_i) \rfloor$ may differ from v when the exact sum lands in the narrow "error region" $\mathbb{Z} + \frac{1}{2} \pm \epsilon$ with $\epsilon = \sum_i \epsilon_i$. Using IEEE 754 doubles, $|\epsilon_i| < 2^{-53}$, hence $|\epsilon| < k\, 2^{-53}$ (e.g., $|\epsilon| < 2^{-48}$ for $k \le 32$). In practice, the following three regimes make the issue benign:

- When $|x| \ll q$ (e.g., $|x| < q/4$, as is guaranteed in the second extension inside key switching and in the scale-down after tensoring, where the extended base is chosen with p substantially larger than q), the exact sum is bounded away from $\mathbb{Z} + \frac{1}{2}$ and the estimate is provably always correct.
- When x is (pseudo)random in $\mathbb{Z}_q$, the probability of landing in the error region is bounded by $2|\epsilon|$, i.e., negligible in practice.
- Even when an off-by-one estimate does occur, in the homomorphic-encryption context it merely contributes q to a value that is subsequently interpreted modulo q or scaled down, adding a practically negligible amount to the ciphertext noise (at most 1–2 bits in the worst case, and unobservable in experiments [HPS19]).

Notably, the estimate imposes no constraint on the sizes of the moduli themselves: every summand $z_i = y_i/q_i$ lies in $[-\frac{1}{2}, \frac{1}{2})$, so the floating-point error is relative to a bounded quantity and independent of $\max_i(q_i)$ — full word-sized moduli are supported with plain doubles. A moduli-size constraint does arise from the floating-point precision, but in the *scaling* procedures, where the fractional constants are multiplied by full-sized residues; we discuss it in Section 5.4.

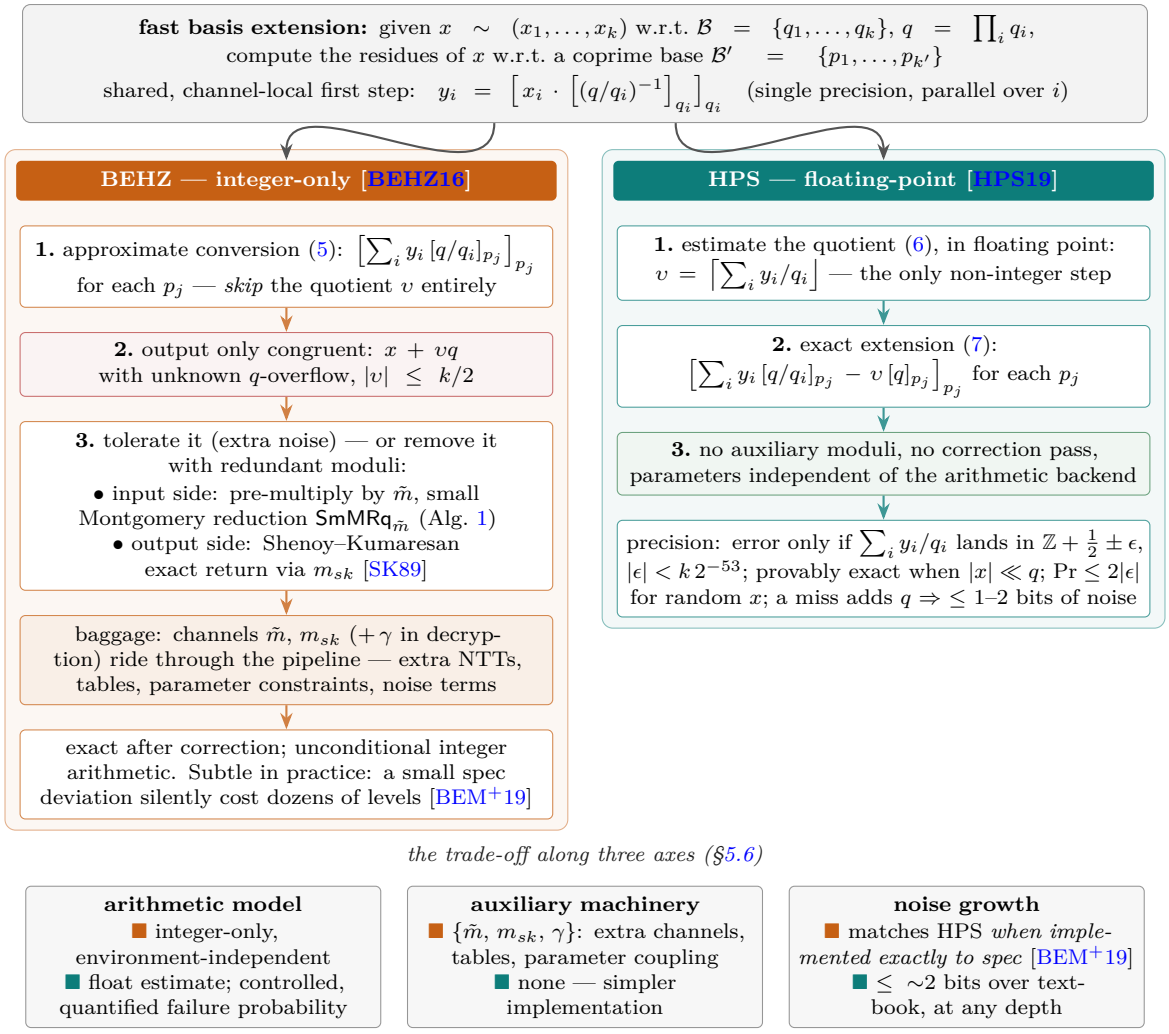


Figure 3: The two realizations of the RNS primitives, shown for fast basis extension (Section 5.3). Both share the channel-local products y_i; they differ in how the reconstruction quotient v of Eq. (3) is handled. BEHZ (left) never computes v: the fast conversion carries a q-overflow that is either tolerated or removed by integer-only corrections, at the price of auxiliary residue channels threaded through the entire pipeline. HPS (right) estimates v directly in floating point, yielding an (almost always) exact conversion with no auxiliary machinery, at the price of a small, quantifiable probability of an off-by-one estimate. The bottom row condenses the comparison of Section 5.6.

5.4 Simple scaling in RNS

The second magnitude-dependent primitive is scaling with rounding: given $x \in \mathbb{Z}_q$ in CRT representation and an integer $t \geq 2$ (with $t \ll q$), compute

$$y \;=\; \left[\left\lceil \tfrac{t}{q} \cdot x \right\rfloor\right]_t. \tag{8}$$

This is the core of BFV decryption, and the same pattern (division by a subproduct of the modulus, followed by rounding) appears in modulus switching and CKKS rescaling.

5.4.1 HPS simple scaling

Starting from Eq. (4) and multiplying by t/q,

$$\left\lceil \tfrac{t}{q}\, x \right\rfloor = \left\lceil \sum_{i=1}^{k} x_i \cdot \left[\left(\tfrac{q}{q_i}\right)^{-1}\right]_{q_i} \cdot \frac{t}{q_i} \right\rfloor - \upsilon' t \quad\Longrightarrow\quad y = \left[\left[\sum_{i=1}^{k} x_i \cdot \left[\left(\tfrac{q}{q_i}\right)^{-1}\right]_{q_i} \cdot \frac{t}{q_i} \right] \right]_t, \tag{9}$$

since the term $\upsilon' t$ vanishes modulo t. The rational constants are precomputed and split into integral and fractional parts,

$$\frac{t \cdot \left[(q/q_i)^{-1}\right]_{q_i}}{q_i} \;=\; \omega_i + \theta_i, \qquad \omega_i \in \mathbb{Z}_t, \quad \theta_i \in [-\tfrac{1}{2}, \tfrac{1}{2}),$$

with ω_i stored as single-precision integers and θ_i as (extended) doubles. At runtime one computes $w := \left[\sum_i x_i\, \omega_i\right]_t$ in integer arithmetic and $v := \lceil \sum_i x_i\, \theta_i \rfloor$ in floating-point arithmetic, and outputs $[w + v]_t$. The precision analysis mirrors Section 5.3.2; in decryption the ciphertext noise guarantees the pre-rounding value is within $1/4$ of an integer, so restricting $q_i \leq 2^{\nu-1}/k$ (ν being the significand precision in bits) ensures exactness (this tightens the textbook correctness bound by at most one bit, from $(\Delta - r_t(q))/2$ to $(\Delta - r_t(q))/4$, where $\Delta = \lfloor q/t \rfloor$ and $r_t(q) = q - t\Delta$).

Unlike in basis extension (Section 5.3.2), the floating-point precision thus constrains the size of the RNS moduli: here the fractional constants θ_i, stored with absolute error $\sim 2^{-\nu}$, are multiplied by full-sized residues x_i, so the error budget scales with $\max_i(q_i)$. For decryption-type scaling with double precision ($\nu = 53$), the requirement $q_i \leq 2^{\nu-1}/k$ limits the usable moduli for signed-integer modular arithmetic; extended double precision ($\nu = 65$, the x86 `long double`) supports moduli up to 59 bits, and software `double double` ($\nu = 113$) supports the full 60-bit range at a significant performance cost [BPA+21]. In the original HPS formulation, this trade-off between ν, $\max_i(q_i)$, and k was a distinctive engineering constraint of the approach: HPS was fastest exactly when native floating-point types sufficed. This constraint was subsequently removed by Kim, Polyakov, and Zucca [KPZ21], who reformulated the HPS scaling procedures — by decomposing the input residues into smaller digits before the floating-point summation — so that the full range of word-sized CRT moduli (up to 60 bits) is supported using regular double-precision arithmetic alone, without any resort to extended or software-emulated precision.

5.4.2 BEHZ simple scaling

BEHZ replaces rounding by *exact flooring*, which is compatible with integer-only RNS:

$$\left\lfloor \tfrac{t}{q}\, [x]_q \right\rfloor \;=\; \frac{t\,[x]_q - |t\,x|_q}{q},$$

an exact integer division. Modulo t the first term cancels, and $-|t\,x|_q/q$ is obtained via FastBaseConv from base q to t. Because the fast conversion is inexact, a redundant modulus γ is introduced: the computation is carried out simultaneously modulo t and modulo γ, and the γ-channel is used both to cancel the q-overflow and to correct the flooring-vs-rounding discrepancy. The auxiliary channel slightly reduces the tolerable noise: correctness requires $\|e\|_\infty \le \Delta(\frac{1}{2} - \frac{k}{\gamma}) - \frac{|q|_t}{2}$, approaching the textbook bound only as $\gamma \gg k$.

5.5 Complex scaling: scaling out of an extended base

The most demanding pattern combines both primitives. After a tensor-style multiplication the intermediate value lives in an extended base: $x \in \mathbb{Z} \cap [-qp/2t,\ qp/2t)$ represented in $\mathcal{B} \cup \mathcal{B}' = \{q_1, \ldots, q_k, p_1, \ldots, p_{k'}\}$, and one must compute

$$y \;=\; \Big[\Big\lceil \tfrac{t}{q} \cdot x \Big\rfloor\Big]_q,$$

i.e., scale by t/q *and* land back in the original base. Below we denote the composite modulus by $Q := qp$; all precomputed constants are written explicitly in terms of $Q/q_i = (q/q_i)\,p$ and $Q/p_j = q\,(p/p_j)$. In the HPS formulation the complex scaling is accomplished in two steps:

1. **Scale within the complementary base.** Apply simple scaling with $q' = qp$ and $t' = tp$, producing $y' := \big[\lceil t/q \cdot x \rfloor\big]_p$ directly in the base $\mathcal{B}'$. (Because $|x| < qp/2t$, the scaled value lies in $[-p/2, p/2)$, so the mod-p representation determines y' exactly, without modular reduction.) Concretely, with precomputed splits

$$\frac{t \cdot \big[(Q/q_i)^{-1}\big]_{q_i} \cdot p}{q_i} \;=\; \omega_i' + \theta_i', \qquad \omega_i' \in \mathbb{Z}_p,\quad \theta_i' \in [-\tfrac{1}{2}, \tfrac{1}{2}),$$

   and constants $\lambda_j := \big[\, t \cdot \big[(Q/p_j)^{-1}\big]_{p_j} \cdot (p/p_j)\,\big]_{p_j}$, one computes $v := \lceil \sum_i \theta_i' x_i \rfloor$ once, and $y_j' := \big[\lambda_j x_j' + \sum_i [\omega_i']_{p_j}\, x_i + v\big]_{p_j}$ for each j.

2. **Extend back.** Apply FastBaseExt (Section 5.3.2) to extend y' from $\mathcal{B}'$ to $\mathcal{B}$, then discard the $\mathcal{B}'$ components. Because $|y'| \le q/2 \ll p/2$, this second extension provably never lands in the floating-point error region — exactness comes for free from the parameter choice $p > q$.

Rounding here need not be exact: a rounding error simply adds to the ciphertext noise, in contrast to decryption where exactness is required.

The BEHZ counterpart follows the same blueprint but with integer-only tools: lifting via FastBaseConv with input-side pre-multiplication by $\tilde{m}$ and small Montgomery reduction (Algorithm 1) to remove q-overflows; *approximate flooring* in the complementary base; and an exact return conversion via the Shenoy–Kumaresan technique using the carried modulus m_{sk}. The auxiliary channels $\{\tilde{m}, m_{sk}\}$ ride along through the tensor product, adding NTTs and modular multiplications throughout.

5.6 BEHZ versus HPS: a summary

The two families realize the same primitives with different tools, and the contrast can be summarized along three axes. First, the *arithmetic model*: BEHZ is integer-only, which makes its

correctness unconditional and its behavior independent of the floating-point environment; HPS relies on floating-point estimation of the reconstruction quotient, trading a small, controllable approximation-error probability for the elimination of all auxiliary moduli and correction passes, but tying the supported moduli sizes, in the scaling procedures of its original formulation, to the available floating-point precision (a limitation later lifted by [KPZ21], as discussed in Section 5.4). Second, the *auxiliary machinery*: BEHZ requires the redundant moduli $\tilde{m}$, m_{sk} (and γ for decryption), which add residue channels, precomputed tables, and parameter-selection constraints; HPS requires none, which makes it simpler to implement and its parameter generation independent of the arithmetic backend. Third, *noise behavior*: the HPS procedures add at most $\sim$2 bits of noise relative to the textbook scheme for any depth, and the worst-case noise analyses of the two variants are essentially equivalent. The initial experimental comparison of [BPA+21] observed a markedly faster practical noise growth for BEHZ — the multiplicative depth reached by HPS was typically 15–30% larger for the same parameters — and attributed it to correlations introduced by the fast-conversion overflow term vq, which would invalidate the average-case (Central Limit Theorem) noise heuristics. A subsequent note by Bajard et al. [BEM+19] identified the actual cause: the implementations of the small Montgomery reduction procedure $\mathsf{SmMRq}_{\tilde{m}}$ (Algorithm 1) in SEAL and PALISADE deviated slightly from the original BEHZ specification. Once this discrepancy is fixed, both variants achieve the same multiplicative depth in practice, and the average-case noise behavior of a correctly implemented BEHZ matches that of HPS. The episode is instructive in its own right: the correction machinery of the integer-only approach is subtle enough that a small implementation deviation, invisible to worst-case analysis, silently cost dozens of levels of multiplicative depth — reinforcing the implementation-simplicity argument for HPS even as it removes the noise-growth argument against BEHZ. A detailed experimental comparison of both variants on CPUs and GPUs is given in [BPA+21], with the noise correction in [BEM+19].

Subsequent work by Kim, Polyakov, and Zucca [KPZ21] further improved the HPS procedures: beyond removing the floating-point precision constraint (Section 5.4), it reduced the noise growth of the multiplication pipeline and extended the same RNS toolkit to an improved BGV instantiation, unifying the treatment of the exact schemes.

These trade-offs explain the library adoption pattern traced in Section 4: SEAL's first Full RNS BFV used BEHZ, PALISADE adopted HPS (with BEHZ added for the experimental comparison of [BPA+21]), and OpenFHE implements the improved HPS-style variants of [KPZ21] as its default.

6 Composed RNS Procedures Across Schemes

The basis-extension and scaling primitives of Section 5 are not BFV-specific. In this section we show how they compose into higher-level procedures that appear in every vectorized RLWE scheme, such as modulus switching and key switching. Throughout, the modulus is a chain $Q_\ell = \prod_{i=0}^{\ell} q_i$ of NTT-friendly primes, and ciphertexts at level ℓ live in $\mathcal{R}^2_{Q_\ell}$.

6.1 Modulus switching and CKKS rescaling

Modulus switching maps a ciphertext modulo Q_ℓ to a ciphertext modulo $Q_{\ell-1} = Q_\ell/q_\ell$ encrypting (approximately) the same plaintext, by computing $c' = \left\lceil \frac{Q_{\ell-1}}{Q_\ell}\, c \right\rfloor = \lceil c/q_\ell \rfloor$. This operation is a special case of the simple scaling described in Section 5.4. In BGV, modulus switching controls

noise; in BFV, modulus switching is used to compress a ciphertext result before sending it to the decryptor or to reduce the computational complexity of subsequent operations; in CKKS, the identical operation, called *rescaling*, additionally reduces the encrypted scale, implementing fixed-point precision management. A key observation makes this operation lightweight in RNS: because q_ℓ is itself one of the RNS moduli, no generic scaling procedure is needed. Writing the division with explicit rounding error,

$$c' = \frac{c - [c]_{q_\ell}}{q_\ell}, \qquad \text{i.e.,} \qquad c'_i = \left[q_\ell^{-1} (c_i - [c]_{q_\ell}) \right]_{q_i} \quad \text{for } i < \ell, \tag{10}$$

one needs only the residue $[c]_{q_\ell}$ (one inverse NTT in the dropped channel if the ciphertext is kept in evaluation representation), its broadcast into the remaining channels, and one modular multiplication by the precomputed $[q_\ell^{-1}]_{q_i}$ per channel. This is the RNS-friendly rescaling introduced for CKKS by Cheon, Han, Kim, Kim, and Song [CHK+18], and the analogous BGV modulus switching (with an extra correction added to preserve the plaintext value) used since HElib [GHS12].

The elegance of Eq. (10) comes with a structural cost that is easy to overlook: *CKKS rescaling granularity is now tied to the modulus chain.* One can only divide by (products of) the primes actually present in the chain; rescaling by an arbitrary small amount — say, 10 bits — is impossible unless the chain was explicitly designed with such primes. In CKKS this couples numerical-precision management to modulus-chain design and makes parameter selection more rigid. It also introduces a persistent source of *scaling error*: since NTT primes can only approximate the target scaling factor ($q_\ell \approx 2^\Delta$ but never exactly), every rescaling in the original Full RNS variant [CHK+18] perturbs the encrypted scale, and the accumulated deviation manifests as additional approximation error in the decrypted results — an error inherent to the RNS formulation and absent from the multiprecision CKKS, where the scaling factor can be an exact power of two. Three lines of work have addressed this limitation.

Eliminating the scaling error. Kim, Papadimitriou, and Polyakov [KPP22] showed that the scaling error of the RNS variant can be eliminated outright, without changing the modulus chain. Instead of pretending that every level shares a common scaling factor Δ, their reduced-error CKKS variant maintains an exact, level-dependent scaling factor that tracks the primes actually consumed — after a multiplication at scale Δ_ℓ is rescaled by q_ℓ, the scale becomes $\Delta_{\ell-1} = \Delta_\ell^2 / q_\ell$ — and adjusts subsequent operations (automatically inserting scalar corrections when operands at different scales meet) so that all arithmetic remains exact with respect to the tracked scale. Combined with a reduced-error encryption procedure that pushes the fresh LWE noise below the significant bits of the message, this recovers several bits of precision (roughly 4 bits for uniform ternary secrets, and up to 12 bits relative to prior RNS implementations, depending on the setting) at essentially no performance cost. This automated scaling-factor management is implemented in OpenFHE as its `FLEXIBLEAUTO` and `FLEXIBLEAUTOEXT` rescaling modes, which are the library defaults. In HElib and Lattigo, similar exact-scale tracking mechanisms are not a selectable mode but the built-in behavior of the evaluator.

Scaling factors beyond the word size: composite scaling. The precision achievable with the above techniques is still capped by the machine word: a single-prime scaling factor cannot exceed the word size W, which is limiting on 32-bit hardware accelerators (GPUs, ASICs) and for high-precision applications. Composite scaling [AAB+23] lifts this cap by composing each

scaling factor from $d \geq 2$ smaller NTT primes with $\prod_{j=1}^{d} q_{\ell,j} \approx 2^{\Delta}$; rescaling drops d primes from the chain at once, exactly as in Eq. (10) applied iteratively. This makes scaling factors of, e.g., 64 or more bits available on architectures whose native word is smaller, and it composes with the reduced-error exact scale tracking. The authors prove and experimentally confirm functional equivalence with single-prime scaling; the technique is integrated in OpenFHE (as composite scaling modes) and is also implemented in Lattigo.

Decoupling scale from modulus: grafting. Grafting [CCK+25] attacks the root cause — the rigid coupling between the scale factors (a numerical-precision concern) and the RNS factors of the modulus (a cryptographic one). The ciphertext modulus is assembled from reusable, word-size-optimized factors called *sprouts*, including power-of-two factors that need not be NTT primes; rescaling by an arbitrary number of bits is then realized as a modulus-switching step (Section 6.1) rather than by dropping designated chain primes, since consecutive moduli are no longer divisible by one another. This decoupling removes both the scaling error and the NTT-prime scarcity problem (large chains of small primes for a given ring dimension), and improves performance by packing the modulus into full machine-word limbs — fewer, denser RNS channels for the same modulus size. Grafting has been adopted in HEAAN-family implementations and in recent GPU libraries.

Taken together, these three refinements illustrate a recurring pattern of the Full RNS era: the residue representation first trades exactness or flexibility for speed, and subsequent work recovers the lost property — exact scales, large scaling factors, arbitrary rescaling — by more careful bookkeeping on top of the same residue-level primitives. These refinements occupy the rightmost band of Figure 1.

6.2 Key switching

Key switching converts a ciphertext component decryptable under a key s' (e.g., s^2 after multiplication, or a rotated key after an automorphism) into one decryptable under s. All modern constructions share the same skeleton: decompose the input c into low-norm digits $\{c^{(j)}\}$ whose recombination is linear, take an inner product with key-switching keys that encrypt the correspondingly weighted multiples of s', and, in modern variants, scale the result back down. The decomposition exists solely to control the noise term $\sum_j \langle e_j, c^{(j)} \rangle$: smaller digits mean less noise but more NTTs and larger keys.

The evolution of key switching methods included the following milestones.

(a) **Positional digit decomposition (BV).** Brakerski and Vaikuntanathan proposed to decompose each coefficient in a radix w: $c = \sum_j c^{(j)} w^j$ [BV11]. In RNS form this is exactly pattern (iii) of Section 5.2: extracting base-w digits of a multi-hundred-bit integer is a positional operation requiring full CRT reconstruction — precisely what RNS is meant to avoid.

(b) **Auxiliary-modulus raising (GHS).** Avoid decomposition entirely by lifting c from $\mathcal{R}_Q$ to $\mathcal{R}_{QP}$ for a special modulus $P > Q$, multiplying by a key-switching key defined modulo QP, and scaling back down by P [GHS12]. The noise added by the key is divided by P and becomes negligible, but requiring $P > Q$ doubles the effective modulus, which at fixed security halves the usable ciphertext modulus budget. A hybrid BV-GHS method was also proposed in [GHS12].

(c) **BV RNS digit decomposition.** Bajard et al. [BEHZ16] observed (with a refinement in [HPS19]) that the CRT decomposition itself can serve as the digit decomposition: the map $c \mapsto ([c]_{q_1}, \ldots, [c]_{q_k})$ is linear via Eq. (3)-style recombination with weights $\left[(q/q_i)^{-1}\right]_{q_i} \cdot (q/q_i)$, its digits $[c]_{q_i}$ are available *for free* in an RNS representation, and their norms are bounded by $q_i/2 \approx w/2$. The evaluation key is simply generated with respect to the weights $\left[\left[(q/q_i)^{-1}\right]_{q_i} \cdot (q/q_i) \cdot s'\right]_Q$ (equivalently, keys that are $[s']_{q_i}$ in channel i and 0 elsewhere), eliminating both the decomposition cost and, in the refined formulation, all precomputed decomposition constants.

(d) **Hybrid key switching in RNS.** The hybrid method originally proposed in [GHS12] was extended to CKKS in its modern RNS form by Han and Ki and adapted to BGV/BFV by Kim, Polyakov, and Zucca [HK20, KPZ21]. This method combines (b) and (c) and is the default in OpenFHE, SEAL (for CKKS/BGV), Lattigo, and HEAAN. We describe it in more detail below.

6.2.1 Hybrid key switching in RNS

The RNS base of Q_ℓ is partitioned into d groups ("digits") $\mathcal{D}_1, \ldots, \mathcal{D}_d$ with partial products $\hat{Q}_j$, and a special modulus $P = \prod_j p_j$ is chosen with $P \gtrsim \max_j \hat{Q}_j$ — only as large as one digit, not as large as all of Q_ℓ. The procedure, stated for an input $c \in \mathcal{R}_{Q_\ell}$ and illustrated step by step in Figure 4, is:

1. **Decompose.** Split the RNS representation of c into the d digit groups $c^{(j)} = \left([c]_{q_i}\right)_{q_i \in \mathcal{D}_j}$. Although the splitting is free, NTTs are typically required to convert the input in evaluation representation to the coefficient representation before extracting the digits.

2. **ModUp (basis extension).** Extend each digit from its base $\mathcal{D}_j$ to the full base $\mathcal{B}_{Q_\ell} \cup \mathcal{B}_P$ using fast basis extension (Section 5.3) — d invocations of $\mathsf{FastBaseExt}$ (or $\mathsf{FastBaseConv}$ with or without corrections). This is where the magnitude of each digit, an integer bounded by $\hat{Q}_j/2$, is re-encoded in the larger dynamic range $Q_\ell P$. The extended digits are then converted to the evaluation representation using additional NTTs.

3. **Inner product.** Multiply the raised digits by the key-switching keys $\mathsf{ksk}_j \in \mathcal{R}^2_{Q_\ell P}$, which encrypt $\left[P \cdot \left[(Q_\ell/\hat{Q}_j)^{-1}\right]_{\hat{Q}_j} \cdot (Q_\ell/\hat{Q}_j) \cdot s'\right]_{Q_\ell P}$, and accumulate, entirely component-wise in evaluation representation.

4. **ModDown (complex scaling).** Scale the accumulated result by $1/P$ with rounding and return to the base $\mathcal{B}_{Q_\ell}$: $\mathsf{ModDown}(x) = \left[\lceil x/P \rfloor\right]_{Q_\ell}$, an instance of the complex scaling procedure of Section 5.5 with the roles of the two bases exchanged; since the result is needed only in $\mathcal{B}_{Q_\ell}$ — here the complementary base — the first step already produces it, and the concluding extension back is not required. The key noise is attenuated by the factor P. NTTs are typically required to convert the result back to evaluation representation.

The parameter d interpolates between the extremes: $d = k$ (one prime per digit) recovers pure RNS decomposition with minimal P and maximal NTT count, while $d = 1$ recovers the GHS method with minimal computation and maximal P. In practice d is a tuning knob balancing key size, noise budget, and the $O(d \cdot (k + k'))$ NTTs that dominate key-switching time. Viewed through the lens of this survey, hybrid key switching is nothing more than a careful composition of the two magnitude-dependent RNS primitives — d basis extensions up and one complex scaling

down — wrapped around cheap residue-wise inner products. Its efficiency is inherited directly from the efficiency of those RNS building blocks and NTTs required to support polynomial multiplications in the inner product and operations preceding/following the invocation of key switching.

6.3 BFV-specific composition

For completeness, we note how the same blocks assemble into BFV homomorphic multiplication, the setting in which BEHZ and HPS were originally formulated. Tensoring two ciphertexts requires products without wraparound, so ciphertexts are first lifted from $\mathcal{B}_q$ to $\mathcal{B}_q \cup \mathcal{B}_p$ (basis extension; 4 conversions), the tensor products are computed in the extended base ($qp \gtrsim$ the largest tensored coefficient), the three resulting components are scaled by t/q back into $\mathcal{B}_q$ (complex scaling; 3 invocations), and the quadratic component is relinearized via RNS-digit key switching. Decryption is a single simple-scaling invocation. The cost of the full pipeline is exactly the sum of these building-block costs, which is why the BEHZ/HPS comparison of Section 5.6 transfers to any scheme built from the same blocks.

7 GPU Acceleration of RLWE-based FHE

The dense polynomial arithmetic inherent in modern FHE schemes exposes high degrees of parallelism, which makes encrypted computation amenable to hardware acceleration. In the Full RNS setting, most operations decompose into independent work across polynomial coefficients, RNS limbs, and even multiple ciphertexts in batched workloads. This structure maps naturally onto GPUs: modular additions and multiplications are data-parallel, NTTs decompose into parallel butterfly stages, and the basis-extension and ModDown kernels of key switching consist of large amounts of regular, matrix-like arithmetic (the inner products of Eqs. (5) and (7) across residue channels). Early evidence came from the BFV setting, where a GPU implementation of the HPS variant achieved homomorphic multiplication in 51 ms at multiplicative depth 98 and 128-bit security — two orders of magnitude faster than CPU results [BPA+21] — with the HPS variant consistently outperforming BEHZ by 1.1–1.3× on GPUs, where its avoidance of auxiliary residue channels pays off directly in fewer kernels and smaller constant-memory footprints.

Arithmetic throughput is not the only advantage that matters: modern FHE workloads are often limited by memory bandwidth and data movement as much as by computation. GPUs provide large register files, high-throughput GDDR/HBM memory, and increasingly large on-chip caches. FIDESlib [ADOVLG+25], a state-of-the-art GPU-accelerated Full RNS CKKS library, illustrates that GPUs alleviate the bandwidth bottleneck of CKKS relative to DDR-based CPUs. These considerations place GPUs in an important middle ground between CPUs and custom ASICs.

Among vectorized FHE schemes, CKKS has seen the most GPU-acceleration effort, due both to its popularity for machine learning and to its relatively efficient bootstrapping compared to BFV and BGV, which require costly digit-decomposition subroutines. Phantom [YSD+24] supports accelerated leveled CKKS (as well as BGV and BFV), while Cheddar [CKA26] supports fully bootstrappable CKKS, achieving bootstrapping in ∼22 ms and encrypted ResNet-20 inference in ∼720 ms on an NVIDIA RTX 5090. More recent efforts explore multi-GPU acceleration: FIDESlib supports multi-GPU contexts, and Cerium [JKS+25] evaluates a CKKS bootstrap in under 8 ms across 8 NVIDIA B200 GPUs, as well as an encrypted Llama3-8B inference in 134 s. The state-of-the-art GPU results for private LLM inference are presented in [AAPV26].

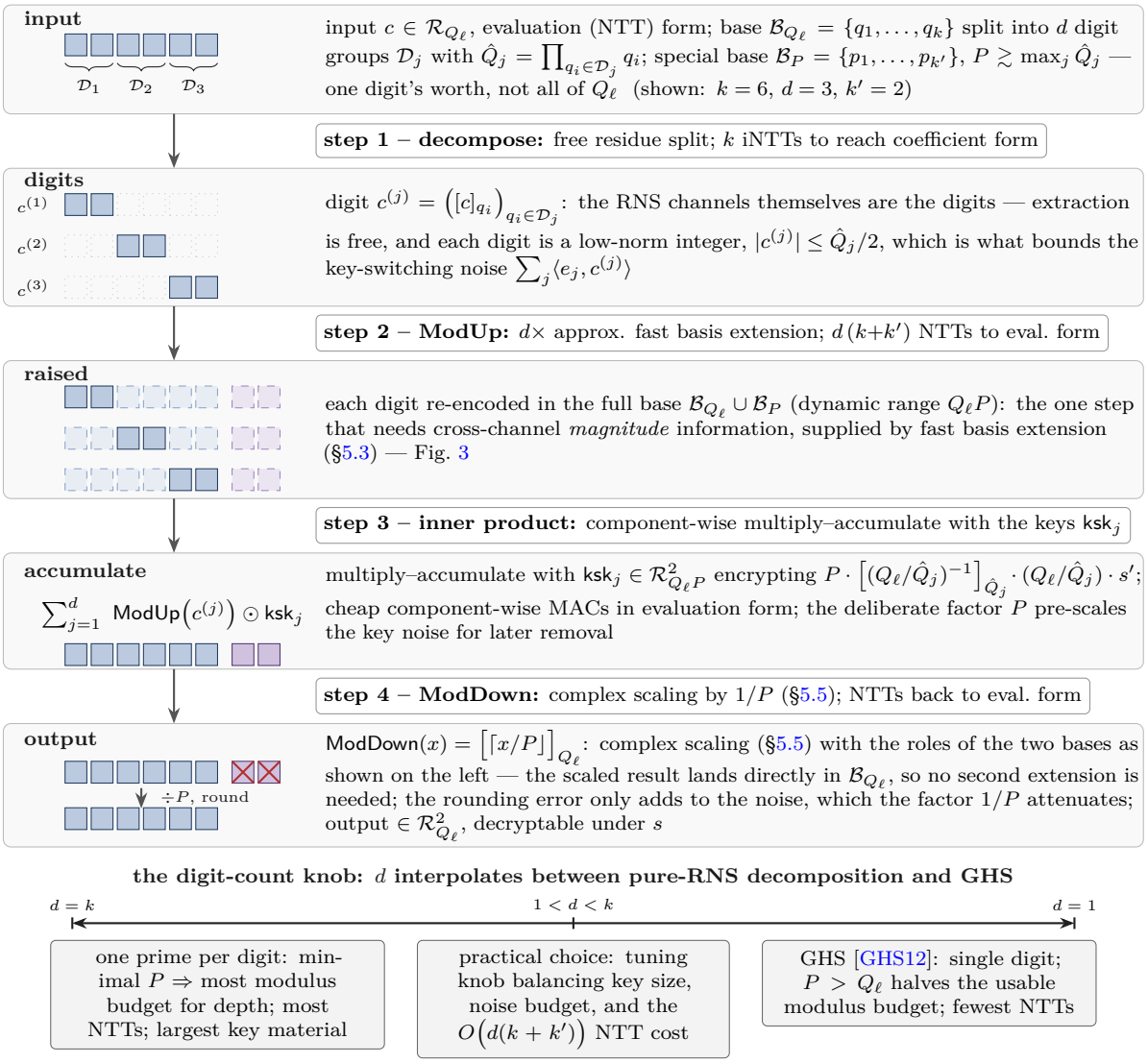


Figure 4: Hybrid key switching in RNS (Section 6.2), drawn for $k = 6$ moduli, $d = 3$ digits of two primes each, and $k' = 2$ special primes. Small squares are RNS limbs: **blue** for $\mathcal{B}_{Q_\ell}$, **purple** for the special base $\mathcal{B}_P$; dashed limbs are produced by fast basis extension. The pipeline composes the two magnitude-dependent primitives of Section 5 — d basis extensions up (ModUp), one complex scaling down (ModDown) — around cheap residue-wise inner products, with the $O\big(d(k+k')\big)$ NTT conversions dominating the cost. The slider (bottom) shows the digit count d interpolating between pure RNS decomposition ($d = k$) and the GHS method ($d = 1$).

8 A Renewed Interest in Positional Arithmetic

The narrative of this survey has been one of a decisive migration away from positional (multiprecision) representations toward Full RNS, motivated by the poor performance of arbitrary-precision arithmetic on the magnitude-dependent operations. It is therefore notable that a recent line of work revisits positional — that is, non-RNS — representations of the large coefficients, arguing that some of the structural costs of Full RNS are avoidable if the big-number arithmetic is decoupled from the cyclotomic arithmetic rather than fused with it through a shared NTT-friendly prime basis.

The starting observation is that the Full RNS design ties two distinct concerns together: the decomposition of a large coefficient into machine-word pieces (needed for arithmetic and for the digit decomposition of key switching) and the number-theoretic transform used for polynomial multiplication. In Full RNS both are carried by the same family of primes, which forces every modulus to be NTT-friendly, makes rescaling granularity a function of the prime sizes (Section 6.1), and — because basis conversion between k-prime bases is a quadratic operation — makes the number of NTTs in key switching scale quadratically in the number of limbs.

Belorgey, Carpov, Gama, Guasch, and Jetchev [BCG+24] showed that these two mechanisms can be fully decoupled. Building on the second gadget decomposition of Kim et al. [KLSS23], which shifts the FHE bottleneck away from the NTT, they represent big numbers positionally, as a bivariate polynomial in which one variable carries the cyclotomic structure and the other carries a base-2^{-K} digit expansion of each coefficient over the *Torus* $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ — the additive group of real numbers modulo 1, familiar as the plaintext and ciphertext domain of TFHE-style schemes. Any coefficient in $\mathbb{Z}_q$ embeds into $\mathbb{T}$ as the fraction x/q, and a base-2^{-K} expansion of a Torus element is simply its fixed-point binary representation, so the digit decomposition needed for key switching is available directly from the representation, without any basis conversion. This makes the number of transforms in the half external product *linear* rather than quadratic in the number of limbs, directly targeting the dominant cost of key switching, while recovering bit-granular rescaling by simple bit shifts. The open-source `poulpy` library [BDM+] implements this bivariate representation across a hardware abstraction layer, reporting the associated benefits: parameterization by a single modulus bit-size rather than a hand-chosen prime chain, a common Torus plaintext space that eases scheme switching in the spirit of CHIMERA [BGGJ20], and a flat, vectorization-friendly memory layout amenable to hardware acceleration. A CKKS evaluator (`poulpy-ckks`) exists as an early, experimental prototype.

These developments have so far been demonstrated primarily in the exact and Boolean settings (Module-LWE, TFHE/FHEW-style bootstrapping), with support for approximate arithmetic still at an early, experimental stage. An interesting open question is whether positional representations can be applied to CKKS as effectively, and how they would compare with the Full RNS variants that dominate approximate homomorphic encryption today. The tension is instructive: the very properties that positional arithmetic recovers — arbitrary, bit-granular rescaling and a scaling factor decoupled from the modulus factors — are precisely those that the RNS refinements of Section 6.1 (reduced-error scaling, composite scaling, grafting) work to reconstruct from within the residue world. Whether the positional approach ultimately offers a more favorable balance for CKKS — weighing its linear-limb key switching and native precision management against the highly optimized, SIMD- and GPU-friendly RNS backends and their mature bootstrapping implementations — is, at the time of writing, an open and only exploratory direction. It is a fitting illustration of this survey's central theme: rather than a single optimal representation, the field continues to rebalance the same fundamental trade-off between parallelism, precision

management, and implementation complexity as hardware and application demands change.

9 Conclusion

The evolution of large-integer arithmetic in FHE libraries is best understood not as a progression toward a single optimal solution, but as a sequence of engineering trade-offs. Each generation of implementations addressed the primary limitations of its predecessors while introducing new architectural constraints and implementation complexity. Arbitrary-precision arithmetic provided a simple and mathematically natural implementation model but suffered from poor performance. CRT-based representations dramatically accelerated polynomial arithmetic while retaining a hybrid architecture whose remaining bottleneck — the magnitude-dependent operations — resisted residue-wise treatment. The Full RNS generation resolved this bottleneck through two primitives, RNS fast basis extension and scaling, realized in two contrasting styles: the integer-only, correction-based BEHZ approach and the floating-point, estimation-based HPS approach. These primitives are crucial for every performance-critical procedure in modern libraries, from CKKS rescaling to hybrid key switching and bootstrapping, on CPUs and GPUs alike — although at the cost of increasingly sophisticated basis management, modulus-chain design, and memory-efficient data movement.

As FHE continues to evolve alongside modern hardware architectures and increasingly demanding AI workloads, these trade-offs remain highly relevant. Rather than viewing existing approaches as superseding one another, future implementations may benefit from revisiting alternative arithmetic backends whenever they offer a more favorable balance between performance, flexibility, and implementation complexity. Understanding these trade-offs is therefore essential not only for interpreting the evolution of existing FHE libraries but also for designing the next generation of efficient homomorphic encryption software.

Acknowledgements. The authors acknowledge the use of Claude for polishing and proofreading the text in the paper. The authors have reviewed the changes and assume full responsibility for the final manuscript.

References

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

[AAB+23] Rashmi Agrawal, Jung Ho Ahn, Flavio Bergamaschi, Rosario Cammarota, Jung Hee Cheon, Fillipe D. M. de Souza, Huijing Gong, Minsik Kang, Duhyeong Kim, Jongmin Kim, Hubert de Lassus, Jai Hyun Park, Michael Steiner, and Wen Wang. High-precision RNS-CKKS on fixed but smaller word-size architectures: theory and application. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*, pages 23–34. ACM, 2023. https://eprint.iacr.org/2023/1462.

[AAPV26] Ahmad Al Badawi, Andreea Alexandru, Yuriy Polyakov, and Vinod Vaikuntanathan. SoK: Private LLM inference using approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2026/935, 2026.

[ADOVLG+25] Carlos Agulló-Domingo, Óscar Vera-López, Seyda Guzelhan, Lohit Daksha, Aymane El Jerari, Kaustubh Shivdikar, Rashmi Agrawal, David Kaeli, Ajay Joshi, and José L. Abellán. FIDESlib: A fully-fledged open-source FHE library for efficient CKKS on GPUs. In *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2025. https://arxiv.org/abs/2507.04775.

[BCC+24] Jean-Philippe Bossuat, Rosario Cammarota, Ilaria Chillotti, Benjamin R. Curtis, Wei Dai, Huijing Gong, Erin Hales, Duhyeong Kim, Bryan Kumara, Changmin Lee, Xianhui Lu, Carsten Maple, Alberto Pedrouzo-Ulloa, Rachel Player, Yuriy Polyakov, Luis Antonio Ruiz Lopez, Yongsoo Song, and Donggeon Yhee. Security guidelines for implementing homomorphic encryption. *IACR Communications in Cryptology*, 1(4), 2024. https://doi.org/10.62056/anxra69p1.

[BCG+24] Mariya Georgieva Belorgey, Sergiu Carpov, Nicolas Gama, Sandra Guasch, and Dimitar Jetchev. Revisiting key decomposition techniques for FHE: Simpler, faster and more generic. In *Advances in Cryptology – ASIACRYPT 2024*, pages 176–207. Springer, 2024. https://eprint.iacr.org/2023/771.

[BDM+] Jean-Philippe Bossuat, Jules Dumezy, Rasoul Akhavan Mahdavi, Janmajaya Mall, Cedoor, and Luis Ruiz-Lopez. Poulpy: A backend-agnostic modular FHE library over the torus using bivariate polynomial representation. https://github.com/poulpy-fhe/poulpy.

[BEHZ16] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *Selected Areas in Cryptography (SAC)*, pages 423–442. Springer, 2016.

[BEM+19] Jean-Claude Bajard, Julien Eynard, Paulo Martins, Leonel Sousa, and Vincent Zucca. Note on the noise growth of the RNS variants of the BFV scheme. Cryptology ePrint Archive, Report 2019/1266, 2019. https://eprint.iacr.org/2019/1266.

[BGGJ20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. CHIMERA: Combining ring-LWE-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020. https://doi.org/10.1515/jmc-2019-0026.

[BGP+20] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, Kurt Rohloff, and Vinod Vaikuntanathan. Optimized homomorphic encryption solution for secure genome-wide association studies. *BMC Medical Genomics*, 13(Suppl 7):83, 2020. https://doi.org/10.1186/s12920-020-0719-9.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science (ITCS)*, pages 309–325, 2012.

[BHHH19] Flavio Bergamaschi, Shai Halevi, Tzipora T. Halevi, and Hamish Hunt. Homomorphic training of 30,000 logistic regression models. In *Applied Cryptography and Network Security (ACNS)*, pages 592–611. Springer, 2019. https://doi.org/10.1007/978-3-030-21568-2_29.

[BPA+21] Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme. *IEEE Transactions on Emerging Topics in Computing*, 9(2):941–956, 2021. https://eprint.iacr.org/2018/589.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, pages 868–886. Springer, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, pages 505—-524, Berlin, Heidelberg, 2011. Springer-Verlag.

[CCK+25] Jung Hee Cheon, Hyeongmin Choe, Minsik Kang, Jaehyung Kim, Seonghak Kim, Johannes Mono, and Taeyeong Noh. Grafting: Decoupled scale factors and modulus in RNS-CKKS. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2025. https://eprint.iacr.org/2024/1014.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *Selected Areas in Cryptography (SAC)*, pages 347–368. Springer, 2018.

[CKA26] Wonseok Choi, Jongmin Kim, and Jung Ho Ahn. Cheddar: A swift fully homomorphic encryption library designed for GPU architectures. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2026. https://arxiv.org/abs/2407.13055.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437. Springer, 2017.

[DAR] DARPA. PROCEED: PROgramming computation on EncryptEd data. https://www.darpa.mil/research/programs/programming-computation-on-encrypted-data. Accessed: 2026-08-03.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

[Gen09] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology – CRYPTO 2012*, pages 850–867. Springer, 2012.

[Gt] Torbjörn Granlund and the GMP development team. GNU MP: The GNU multiple precision arithmetic library. https://gmplib.org.

[HK20] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Topics in Cryptology – CT-RSA 2020*, pages 364–390. Springer, 2020.

[HPS19] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In *Topics in Cryptology – CT-RSA 2019*, pages 83–105. Springer, 2019. https://eprint.iacr.org/2018/117.

[HS20] Shai Halevi and Victor Shoup. Design and implementation of HElib: A homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020.

[JKS+25] Siddharth Jayashankar, Joshua Kim, Michael B. Sullivan, Wenting Zheng, and Dimitrios Skarlatos. A scalable multi-GPU framework for encrypted large-model inference (Cerium). arXiv preprint arXiv:2512.11269, 2025. https://arxiv.org/abs/2512.11269.

[KJT+20] Tsung-Ting Kuo, Xiaoqian Jiang, Haixu Tang, XiaoFeng Wang, Tyler Bath, Diyue Bu, Lei Wang, Arif Harmanci, Shaojie Zhang, Degui Zhi, Heidi J. Sofia, and Lucila Ohno-Machado. iDASH secure genome analysis competition 2018: Blockchain genomic data access logging, homomorphic encryption on GWAS, and DNA segment searching. *BMC Medical Genomics*, 13(Suppl 7):98, 2020. https://doi.org/10.1186/s12920-020-0715-0.

[KLSS23] Miran Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Accelerating HE operations from key decomposition technique. In *CRYPTO 2023*, pages 70–92, Berlin, Heidelberg, 2023. Springer-Verlag.

[KPP22] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. In *Topics in Cryptology – CT-RSA 2022*, pages 120–144. Springer, 2022. https://eprint.iacr.org/2020/1118.

[KPZ21] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In *Advances in Cryptology – ASIACRYPT 2021*, pages 608–639. Springer, 2021. https://eprint.iacr.org/2021/204.

[Mic] Microsoft Research. Microsoft SEAL (release 4.x). https://github.com/microsoft/SEAL.

[PRR] Yuriy Polyakov, Kurt Rohloff, and Gerard W. Ryan. PALISADE lattice cryptography library. https://palisade-crypto.org.

[Sho] Victor Shoup. NTL: A library for doing number theory. https://libntl.org.

[SK89] A. P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computers*, 38(2):292–297, 1989.

[Tun23] Tune Insight SA. Lattigo v5: Lattice-based multiparty homomorphic encryption library in Go. https://github.com/tuneinsight/lattigo, 2023.

[YSD+24] Hao Yang, Shiyu Shen, Wangchen Dai, Lu Zhou, Zhe Liu, and Yunlei Zhao. Phantom: A CUDA-accelerated word-wise homomorphic encryption library. *IEEE Transactions on Dependable and Secure Computing*, 2024. https://eprint.iacr.org/2023/049.