Revisiting Key Decomposition Techniques for FHE: Simpler, Faster and More Generic

M.G. Belorgey[1], S. Carpov[1], N. Gama[2], S. Guasch[2], D. Jetchev[1]

[1] Inpher
[2] SandboxAQ

Abstract. Ring-LWE based homomorphic encryption computations in large depth use a combination of two techniques: 1) decomposition of big numbers into small limbs/digits, and 2) efficient cyclotomic multiplications modulo $X^N + 1$. It was long believed that the two mechanisms had to be strongly related, like in the full-RNS setting that uses a CRT decomposition of big numbers over an NTT-friendly family of prime numbers, and NTT over the same primes for multiplications. However, in this setting, NTT was the bottleneck of all large-depth FHE computations. A breakthrough result from Kim et al. (Crypto'2023) managed to overcome this limitation by introducing a second gadget decomposition and showing that it indeed shifts the bottleneck and renders the cost of NTT computations negligible compared to the rest of the computation. In this paper, we extend this result (far) beyond the Full-RNS settings and show that we can completely decouple the big number decomposition from the cyclotomic arithmetic aspects. As a result, we get modulus switching/rescaling for free. We verify both in theory and in practice that the performance of key-switching, external and internal products and automorphisms using our representation are faster than the one achieved by Kim et al., and we discuss the high impact of these results for low-level or hardware optimizations as well as the benefits of the new parametrizations for FHE compilers. We even manage to lower the running time of the gate bootstrapping of `TFHE` by eliminating one eighth of the FFTs and one sixth of the linear operations, which lowers the running time below 5.5ms on recent CPUs.

1 Introduction

Homomorphic encryption allows computations on encrypted data without decrypting it. Since the first fully homomorphic encryption (FHE) scheme introduced by Gentry in [21], several improvements, implementations and new designs have been proposed. Most of them are based on the ring version of the Learning-With-Error problem (`RingLWE`) defined in [33]. Among the most popular schemes today are `BFV` [20], `BGV` [7], `CKKS` [13], `FHEW` [19] and `TFHE` [14,15]. Subsequent work [6] has been done on the interoperability of `TFHE` with the other two schemes (`BFV` and `CKKS`).

Most of the basic building blocks of the practical homomorphic encryption schemes (e.g., homomorphic multiplication and relinearization, key switching, automorphisms or bootstrapping) reduce to efficient *homomorphic external products*. The latter are basic operations that are combinations of decompositions of one of the inputs into higher-dimensional tensor with entries of small norms (*gadget decompositions*) and polynomial multiplications in order to control the accumulation of noise in the ciphertexts and ensure correct decryption (see, e.g., [6, Defn.1] and [15, Defn.3.12] for typical examples of external product).

A major line of research has been undertaken on using Residue Number Systems (RNS) in the acceleration of the basic underlying polynomial multiplication operations used for computing external products [22]. The full-RNS approach of [4], originally introduced to accelerate `BFV` operations (see also [25]) and subsequently applied to other schemes too such as `CKKS` [12], allows for manipulating large numbers using smaller machine-size moduli and expressing the homomorphic products (internal products) as well as the gadget decompositions of the `BFV` and `CKKS` in the NTT domain. Subsequent hardware acceleration efforts have been made to accelerate RNS-variants of NTT for GPU architectures [32,37] and FPGA architectures [18,30,34,36].

One can thus conceptually differentiate two important aspects of optimization of FHE operations: 1) efficient large integer arithmetic (frontend aspect) that is addressed via mathematical techniques such as gadget decompositions; 2) cyclotomic arithmetic (backend aspects) - the need for efficient arithmetic on the backend to support the basic cyclotomic operations - the choice of floating-point arithmetic or integer arithmetic on the backend. Unfortunately, in most practical implementations, the two aspects are often coupled together, thus, limiting the flexibility for optimization techniques. For instance, by nature, the full-RNS approach to gadget decomposition requires modular arithmetic operations on the backend and thus, is bound to NTT leaving little room for really fast FFT-favored operations (such as the optical FFT approach undertaken in [28]).

Note that there are essentially two classical approaches to 1): decomposition in base 2^K and CRT decomposition over primes of K bits. Assuming that single elementary operations operate natively over K-bit numbers (e.g., 32-bit or 64-bit integers or floating point numbers with 53-bits of mantissa), the fact that CRT arithmetic is exempt from any carry propagation yields very efficient parallel additions and multiplications of ℓ K-bit numbers in $O(\ell)$ operations, whereas the base-2^K counterpart would struggle between $O(\ell \log \ell)$ and $O(\ell^2)$ multiplications and force these elementary operations in sequential mode. For this reason, a natural choice for homomorphic encryption was to prefer CRT representations for efficient implementations of `CKKS` and `BFV`.

A recent breakthrough has been made by Kim et al. towards decoupling 1) and 2) by an approach based on an auxiliary gadget decomposition that replaces expensive modular arithmetic in the computation of the external product by pure arithmetic with small integers via gadget decompositions based on RNS [26].

Note that the RNS-type approach to 1) often has the following drawbacks:

1. *Arithmetic modulo different primes.* The arithmetic has to be carried out modulo different prime numbers of same size, even if they are selected to be

as *friendly* as possible, numbers modulo which arithmetic is really efficient are rare (e.g., Mersenne primes, Fermat primes). There is also a small loss from hardware aspects such as the necessity to have one hardware multiplier per prime.

2. *Unfeasibility of bit extraction/coarse noise granularity.* Bit extraction is not possible in CRT domain modulo a product of small primes, the best granularity of HE noise levels is those of the primes: e.g., 16 to 40 bits. As explained in the analysis of the Hecate compiler [29] this impacts negatively the optimal use of homomorphic budget for CRT-based `CKKS` - one often needs to re-scale the noise to the nearest available level before attempting a homomorphic product after an operation producing less noise (such as a sum, a small linear combination or a trace).
3. *Sub-optimal memory usage with respect to truncations.* Restrictions from high-precision representations to low-precision representations are not simple truncations of limbs (as in the natural base-2^K representation), but require a separate RNS representation.

The above-mentioned carry-less nature and efficiency of parallel CRT multiplications would have been perfect if most homomorphic operations were actual ring operations. However, internal ring multiplications over large numbers never come standalone: first of all, in a `BFV` or `CKKS` internal product they always come in packs of N (giving a $O(N\ell)$ complexity to multiply two polynomials modulo X^N+1), and second, they are always followed by an external product that comprises at least ℓ NTTs (so at least $O(\ell N(\ell + \log N))$ if we use the latest double gadget decomposition of [26] and $O(\ell^2 N \log N)$ before). In other words, since $N \gg \ell$: typically, N is between 1024 and 65536 and ℓ is lower than 100, the complexity of an internal `BFV` or `CKKS` product remains bounded both asymptotically and practically by the complexity of its underlying external product. This means that as long as we make the external product more efficient, we have some margin to degrade the complexity of the internal multiplication of polynomials up to $O(N\ell^2)$ and still accelerating in almost all cases the running time of `CKKS` and `BFV` operations.

It is thus desirable to extend the auxiliary gadget decomposition approach of 1) to the natural 2^K-base large integer representation to overcome the above drawbacks.

Our contributions.

In this work, this is exactly what we achieve - we show how to adapt the *auxiliary gadget decomposition* of [26] to the natural base-2^K representation of numbers and thus, obtain an external product that is simpler and faster.

Our starting point is a common plaintext representation space for all the practical `RingLWE`-based schemes (`BFV`, `CKKS` and `TFHE`) described in [6], namely the $\mathcal{R}$-module $\mathbb{T}_N[X] := \mathbb{R}[X]/\langle X^N+1\rangle/\mathcal{R}$ where $\mathcal{R} = \mathbb{Z}[X]/\langle X^N+1\rangle$ (also thought of as *formal* polynomials of degree $N-1$ whose coefficients are in the torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$). Inspired by the classical Schoenhage–Strassen algorithm for large integer multiplication based on FFT [35], in Section 3, we propose a bivariate integral polynomial representation with small coefficients of these formal polynomials where the evaluation on $Y = 2^{-K}$ (K is the limb size) yields a sufficiently accurate approximation of the toric coefficients. Introducing the auxiliary variable implicitly corresponds to having the second gadget decomposition: on one hand, it yields the natural decomposition in base 2^K of toric elements (evaluation at $Y = 2^{-K}$); on the other hand, it provides a plenitude of choice of bivariate polynomials to approximate the given toric polynomial. A careful normalization and reduction yields a representative with small coefficients - the bivariate polynomial representation that is used for efficient external product (see Section 3.2), thus making it specifically tuned for `TRLWE` operations.

Since the space of `TRLWE` ciphertexts is $\mathbb{T}_N[X]^2$, we express the external product in terms of multiplications of bivariate polynomials with small integer coefficients. It is this reduction that decouples 1) and 2) above - the multiplication can be performed using either FFT or NTT backend arithmetic which we explain in Section 6, thus, enabling optimal use of different architectures for hardware optimization.

A major advantage of our representation is that it addresses challenges 1–3. Indeed, our representation supports efficient sums, normalizations, reductions modulo $\mathbb{Z}$, left and right shifts in $O(\ell)$ and products in $O(\ell \log \ell)$ (only bilinear expressions are needed, we do not need higher degree terms in FHE operations!), bit-decompositions and bit-extraction. In addition, the *prefix property* of the natural base-2^K representation yields a memory-efficient way to pass from a higher-precision representation to a lower precision representation. It provides a single-bit noise granularity, thus, enabling efficient compositions as proposed in [6]. The asymptotic complexity matches the best available full-RNS algorithms for both external products of `TFHE`, and also internal products of `CKKS` and `BFV`.

The sequential nature of carry propagation is mitigated by the parallelism induced by the fact that the operations in our external product occur in a SIMD manner over N elements each time. We increase the complexities of the internal multiplications from $O(N\ell)$ to $O(N\ell \log \ell)$, keeping them negligible compared to an external product, and hence, due to the speed-up of the latter ones, also accelerate internal products of `CKKS` and `BFV`, so that at the end of the day, by not using any of the CRT of full-RNS techniques, and instead, applying the double-gadget decomposition to old-school base-2^K representations directly, we end up speeding up all the existing homomorphic operations of `TFHE`, `CKKS` and `BFV`.

Our practical experiments and benchmarks support the theoretical evidence: first, Table 1 provides evidence for the number of elementary operations (SIMD products and DFTs) in an external product computation in the context of the various FHE building blocks (or more specifically, in the more efficient approach via `HalfRGSW` introduced in Section 3.2). The experiments demonstrate that for FHE parameters $N = 65536$, $\ell \sim 100$ and $K \sim 20$, the cost of SIMD products largely dominates the cost of DFTs, an already strong indication of the parallelization-friendly nature of our approach. Second, we compare our bivariate representation (using both FFT and NTT) to the approach from [26] to demonstrate that, prior to any parallelization or hardware acceleration efforts, we get a better performance (see Table 5). Finally, in Section 4 we even manage to lower the running time of the gate bootstrapping of `TFHE` by eliminating 12.5% of the FFTs and 16.6% of the linear operations.

Future directions.

One of the most important contributions of our work is the flexibility it provides for future hardware acceleration efforts. Everything that we have done in this work is single-core optimizations.

First of all, the normalization and reduction Algorithm 1 (and its extension presented in the appendix Section B) allows for fast `AVX` and GPU-friendly implementations. The bivariate representation yields several advantages making the parallelization very natural. First, as mentioned, it is directly inspired by the Schoenhage–Strassen algorithm for large integer multiplication based on NTT with complexity $O(n \log n \log\log n)$, where n is the number of binary digits of the inputs [35]. In fact, the case of large integer multiplication is recovered directly from our bivariate representation by setting $N = 1$ via the evaluation map $\mathcal{R}[Y] \to \mathcal{R}$ given by $f(Y) \mapsto f(2^K)$. In this work, we use the opposite exponent $f(Y) \mapsto f(2^{-K})$, whose output is dense in $\mathbb{R}_N[X]$, since we care more about high precision approximations of products between bounded real-valued polynomials. Note that the Schoenhage–Strassen is an NTT version of the DFT-based fast polynomial multiplication. Even if the original Schoenhage–Strassen algorithm is not completely easy to parallelize, the large degree N on the variable X makes our bivariate polynomial multiplication friendlier for parallel architectures.

Second, a recursive version of DFT or more generally, a Cooley–Tukey transform [16] allows for parallelization on both X and Y variables. The well-known *in-place* and *in-order* variants of the Cooley–Tukey transform opens the door for highly optimized memory usage as well. When the degree on Y is too small, the alternative naïve multiplication, or single iteration of Karatsuba algorithms discussed here can also be performed in-place.

As already mentioned, the trade-off of using FFT and NTT can benefit from the various hardware acceleration initiatives such as GPU acceleration [32], FPGA acceleration [30,36], optical computing [28] (particularly favorable for the FFT approach), ASICs [17] as well as the outcome of the DPRIVE program [1], especially the recent work [8].

Finally, our work opens up the possibility for an efficient implementation of the scheme switching framework `Chimera` [6] that allows to mix `TFHE`, `CKKS`, `BFV` arithmetic using continuous homomorphic levels. The original proposal was suggesting the use of large numbers fixed-point arithmetic in order to perform efficient additions, multiplication and bit-extraction. Unfortunately, none of the available representations of big numbers in `GMP`, `MPFR` are sufficiently efficient in practice, and a CRT representation would not allow efficient bit-extraction either, thus leading to the same setbacks as Full-RNS. For these reasons, the early attempts to implement `Chimera` were, thus, slow in practice for multiplicative levels ≥ 3. The efficient multi-precision arithmetic we propose in this work, while being compatible with cyclotomic multiplication, bit extraction, and lazy carry propagation make the whole concept practical.

In conclusion, our approach supports the following general principle: instead of one trying to find a numerical representation (e.g., RNS representation) with a gadget decomposition that is not too convoluted, it is more natural to start from the gadget decomposition (e.g., base-2^K representation) and take the output of the gadget decomposition directly as the numerical representation.

2 Preliminaries

Notation

Let $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ be the real torus. For a ring A (e.g., $A = \mathbb{Z}, \mathbb{R}, \mathbb{C}$), let $A_N[X] := A[X]/\langle X^N + 1\rangle$ be the ring of polynomials modulo $X^N + 1$ with coefficients in A and N a power of 2. In particular, let $\mathcal{R} = \mathbb{Z}_N[X]$, which is also the ring of integers of the cyclotomic field $\mathbb{Q}(\zeta_{2N})$.

Let $\mathbb{T}_N[X] = \mathbb{R}_N[X]/\mathcal{R}$ (a.k.a $\mathbb{R}[X]$ mod $X^N + 1$ mod $\mathbb{Z}$) which we view as an $\mathcal{R}$-module (it has no ring structure). Elements of this module can formally be represented using notations borrowed from polynomials, i.e., $a(X) = \sum_{i=0}^{N-1} a_i X^i$ where $a_i \in \mathbb{T}$. Since the coefficient space $\mathbb{T}$ is not a ring, we cannot evaluate these polynomials over any non-integer value, nor multiply two such elements together. However, the notation allows coefficient-wise addition/subtraction of polynomials, projection from and lifts to the continuous ring $\mathbb{R}_N[X]$, and most importantly, we often refer to the module action of $\mathcal{R}$ on $\mathbb{T}_N[X]$ as an *external multiplication* of a by integer polynomial $u(x) = \sum_{i=0}^{N-1} u_i X^i \in \mathcal{R}$ where $u_i \in \mathbb{Z}$ via the natural Cauchy product formula: $(u \cdot a)(X) = \sum_{i=0}^{2N-2} \left(\sum_{j=0}^{i} u_j \cdot a_{i-j}\right) X^i \bmod X^N + 1$.

We denote by $\|.\|_p$ and $\|.\|_\infty$ the standard norms for scalars and vectors over real field or over the integers. By extension, the norms $\|P(X)\|_p$ and $\|P(X)\|_\infty$ of a real or integer polynomial P are the norms of their coefficient vectors. The norm of an element of $\mathbb{T}_N[X]$ is the norm of its centered lift in $\mathbb{R}_N[X]$ with coefficients in $[-1/2, 1/2)$.

2.1 TRLWE

`TRLWE` encrypts elements of a subset of the $\mathcal{R}$-module $\mathbb{T}_N[X]$ called the plaintext space, and which can be finite (`TFHE`), a discrete subgroup (`BFV`) or a continuous set of small elements (`CKKS`). The ciphertext of μ are of the form $c = (a, b = s \cdot a + \mu + e)$, where $s \in \mathcal{B}$ is the secret key, $a \in_R \mathbb{T}_N[X]$ is uniformly random and the $e \in \mathbb{T}_N[X]$ is chosen randomly from an error distribution with mean zero and suitably chosen standard deviation. Without loss of generality, the keyset $\mathcal{B}$ in this work can refer to binary, ternary keys or small keys. The decryption procedure starts by evaluating the phase function on the ciphertext, that is, by computing $\varphi_s(a, b) := b - s \cdot a$. Since $\varphi_s(a, b) = \mu + e$, the plaintext μ is the mean of the phase (the mean being computed over the random coins in the generation of the noise e during encryption).

- `Parameters`: A security level λ and a minimal noise parameter α.
- `KeyGenTRLWE`(λ, α): A uniformly random binary or ternary key $s \in \mathcal{B} \subset \mathcal{R}$.
- `EncTRLWE`(μ, s, α): Choose a uniform random element $a \in \mathbb{T}_N[X]$ and a small Gaussian error $e \in \mathbb{T}_N[X]$ and return $c = (a, b = s \cdot a + \mu + e)$
- `DecTRLWE`(c, s): Compute the phase $\varphi_s(c)$ and round $\varphi_s(c)$ to the nearest point in the plaintext space (when it is discrete), or returns a close approximation (`CKKS`).

2.2 Approximate gadget decompositions and `TRGSW` ciphertexts

A common challenge with various FHE constructions (e.g., `GSW`) is the control on the accumulation of noise after homomorphic operations. A general technique from lattice-based cryptography to address this problem is a concept known as *flattening gadget* or *gadget decomposition*, that is, a map that transforms a ciphertext into a higher dimensional vector with small $\|.\|_\infty$ while preserving some linear-algebraic properties.

Two classical examples of gadget decompositions are numerical base representations and RNS representations. Since most of the FHE literature uses gadget decompositions on integral values, the decompositions themselves are exact. A notable exception to this principle is `TFHE` where one uses floating point arithmetic and therefore, an approximate gadget decomposition. Below we recall the basic definition (see also [15, Defn.3.6]):

Definition 2.1 (approximate gadget decomposition for `TRLWE` samples). *We say that an algorithm* $\texttt{Decomp}_{H,\beta,\epsilon}$ *is a valid gadget decomposition on gadget* $H \in (\mathbb{T}_N[X]^2)^{2\ell}$*, quality (or* $\|\|_\infty$*-bound),* β *and precision* $\epsilon > 0$ *if for any input* $v \in \mathbb{T}_N[X]^2$*, it outputs an element* $u \in \mathcal{R}^{2\ell}$*,* $\|u\|_\infty \leq \beta$ *such that* $\|u \cdot H - v\|_\infty < \epsilon$*.*

As already mentioned in [15], there is a canonical approximate gadget decompositions coming from numerical base representations for any base B_g:

$$H^T = \begin{pmatrix} B_g^{-1} & \dots & B_g^{-\ell} & 0 & \dots & 0 \\ 0 & \dots & 0 & B_g^{-1} & \dots & B_g^{-\ell} \end{pmatrix}$$

The above notion is important for two main reasons: 1) it allows us to define `TRGSW` ciphertexts; 2) it allows us to define an external product between `TRGSW` ciphertexts and `TRLWE` ciphertexts.

We first recall the `TRGSW` ciphertexts: the `TRGSW` encrypts elements of the ring $\mathcal{R}$ with bounded infinity norm. Intuitively, the idea of the original `GSW` scheme [23] is to encrypt a small plaintext $\mu \in \mathbb{Z}$ into a ciphertext that is a matrix C_μ such that the secret key $\mathbf{s}$ is an approximate eigenvector of C_μ with eigenvalue μ. Such an encryption scheme leads to natural homomorphic addition and multiplication operations that are simply matrix additions and matrix multiplications. The naïve idea does not quite work without gadget decompositions since one cannot control the propagation of the errors in the approximate eigenvalues under homomorphic multiplications. The schemes FHEW and TFHE use a ring-variant of the original scheme, the last one with an approximate gadget decomposition:

Definition 2.2 (TRGSW ciphertexts). *Let $H \in (\mathbb{T}_N[X]^2)^{2\ell}$ be a gadget and let $\mu \in \mathcal{R}$ be a plaintext with bounded ℓ_∞-norm. The space of valid TRGSW ciphertexts for μ as TRGSW$(\mu) := \{Z + \mu H\}$, where each element of $Z \in (\mathbb{T}_N[X]^2)^{2\ell}$ is a valid TRLWE ciphertext of zero.*

2.3 External products, relinearization keys and internal products

For the purposes of the current work, we will need a relatively uniform treatment of the various RLWE-based FHE frameworks, most notably, BFV, CKKS and TFHE, as well as their internal products.

It is explained in [6, §2.5] (as well as the particular discussion of BFV and CKKS in Sections 3 and 4, respectively, of *loc.cit.*) how to uniformize the plaintext spaces for all these schemes and view them as various subgroups of the $\mathcal{R}$-module $\mathbb{T}_N[X]$. Once this is done, the homomorphic internal products of BFV and CKKS are expressed in terms of the following basic primitive, the *external product*, a major operation of interest in the current work:

Definition 2.3 (TFHE external product). *Given a flattening gadget H and an approximate gadget decomposition $\texttt{Decomp}_{H,\beta,\epsilon}$, the* external product *in TFHE is a map*

$$\boxdot \colon \texttt{TRGSW} \times \texttt{TRLWE} \to \texttt{TRLWE}$$

defined by

$$C \boxdot c := \texttt{Decomp}_{H,\beta,\epsilon}(c) \cdot C,$$

where $C \in$ TRGSW(μ_1) and $c \in$ TRLWE(μ_2).

One can show [15, Thm.3.13] that under certain specific noise conditions, the above external product $C \boxdot c$ is a valid ciphertext for $\mu_1\mu_2$.

Once we have defined this primitive, we use it to express the various internal products, the major idea being the concept of *relinearization* and *relinearization keys* - we refer the reader to [6, pp.325–326] for a more detailed explanation. The important point is that by using extra key material (relinearization key), one can express the internal product of the BFV scheme in terms of the above external product. More specifically, if we define[3] the relinearization key as $\texttt{RK} = \texttt{TRGSW}(s)$, then for $\star \in \{\texttt{BFV}, \texttt{CKKS}\}$ the homomorphic internal product $\boxtimes \colon \mathbb{T}_N[X]^2 \times \mathbb{T}_N[X]^2 \to \mathbb{T}_N[X]^2$ of two ciphertexts (a_1, b_1) and (a_2, b_2) of plaintexts μ_1 and μ_2, respectively becomes

$$(a_1, b_1) \boxtimes (a_2, b_2) = (C_1, C_0) - \texttt{RK} \boxdot (C_2, 0), \tag{1}$$

[3] There are many equivalent ways to define the relinearization key. Here, we use an external product and a zero term in the TRLWE ciphertext, which can be propagated in the algorithm and simplified, the original references of Fan–Vercauteren [20] present a key-switching that substitute a key s^2 by s.

where $C_0 = b_1 \otimes_\star b_2$, $C_1 = a_1 \otimes_\star b_2 + a_2 \otimes_\star b_1$, $C_2 = a_1 \otimes_\star a_2$, the important point being that this ciphertext is a valid encryption of the plaintext $\mu_1 \otimes_\star \mu_2$. Here, $\otimes_\star \colon \mathbb{T}_N[X] \times \mathbb{T}_N[X] \to \mathbb{T}_N[X]$ indicates a certain product map (depending on the scheme) whose restriction to the plaintext subgroup of $\mathbb{T}_N[X]$ yields the plaintext product.

For instance, if for $\bullet \in \mathbb{T}_N[X]$, $\widetilde{\bullet} \in \mathbb{R}[X]/\langle X^N + 1\rangle$ denotes the unique lift with coefficients in the interval $[-1/2, 1/2)$ then the BFV product $\otimes_{\texttt{BFV}}$ with plaintext modulo p (Montgomery product) is:

$$\otimes_{\texttt{BFV}} \colon \mathbb{T}_N[X] \times \mathbb{T}_N[X] \to \mathbb{T}_N[X], \qquad (u, v) \mapsto p \cdot \widetilde{u} * \widetilde{v}. \tag{2}$$

Similarly, CKKS plaintext products at input level L_{in} are:

$$\begin{aligned}\otimes_{\texttt{CKKS}} \colon \mathbb{T}_N[X] \times \mathbb{T}_N[X] \to \mathbb{T}_N[X],&\\ (u, v) \mapsto 2^{L_{\text{in}}} \cdot \texttt{Round}_{2^{-L_{\text{in}}}}(\widetilde{u}) * \texttt{Round}_{2^{-L_{\text{in}}}}(\widetilde{v})&\end{aligned} \tag{3}$$

In both formulas for (BFV and CKKS product), the symbol $\cdot$ is the external product by an integer, and $*$ represents multiplication in $\mathbb{R}_N[X]$: the input coefficients are first lifted to the real interval $[-1/2, 1/2)$ and in the second case, also rounded to the nearest exact multiple of 2^{-L}. None of these functions is an actual product over the entire space - one needs to restrict the inputs to the subgroups $p^{-1}\mathcal{R}/\mathcal{R}$ and $2^{-L}\mathcal{R}/\mathcal{R}$ of $\mathbb{T}_N[X]$, respectively to obtain products. Yet, the function $\otimes_{\texttt{BFV}}$ has the property that if u and v are close from i/p and j/p where i and j are integers, their product is close to $(ij \bmod p)/p$, which is handy to encode plaintext arithmetic modulo p. The product $\otimes_{\texttt{CKKS}}$ has the property that if u and v are at distance $\leq 2^{L+1}$ away from $i/2^L$ and $j/2^L$ where i and j are smaller than 2^ρ, then their product is at distance $2^{L-\rho}$ away from $ij/2^L$, which is good to encode fixed-point number arithmetic on ρ-bits numbers.

2.4 Table of symbols, orders of magnitudes

When reading the paper, different complexities shall intervene in the different theorems. It is important to keep in mind the difference in orders of magnitude, as for instance: having an arithmetic in $O(N^2)$ is prohibitive, and we must absolutely stick to $O(N \log N)$, however $O(\ell^2)$ is perfectly realistic in some scenario.

The parameters N, K and $\widetilde{K}$ are set once and for all during parameter and key generation, while L, ℓ, and $\tilde{\ell}$ evolve during a homomorphic evaluation computation, following the noise rate variations: They decrease across homomorphic operations, and are reset to a large value after a bootstrapping.

Variable	Range	Meaning
N	$[2^{10}, 2^{16}]$	Power-of-two polynomial modulus: $X^N + 1$
$K, \widetilde{K}$	$[10 - 60]$	Multi-precision representation of Torus elements consist of K-bit limbs (so machine words have at least $2K$ bits to handle products), and gadget decompositions produce $\widetilde{K}$-bit outputs. $\widetilde{K}$ is in general equal to K, but can be chosen smaller in rare circumstances, since $\widetilde{K}$ intervenes in the noise propagation of external products.
L	$[20 - 2000]$	Targeted number of bits of precision of the multiprecision arithmetic (to handle a RLWE ciphertext whose noise rate is $\alpha \approx 2^{-L}$). For a given cryptographic security parameter, each key dimension is associated to a maximal noise level L: for 128-bit security, count $L_{\max} = 20, 880, 1761$ for respectively $N = 2^{10}, 2^{15}, 2^{16}$
$\ell, \tilde{\ell}, \tilde{\ell}_A$	[1-100]	Number of limbs per coefficient in a RLWE ciphertext (without tilde) or in a gadget decomposition (with tilde, and/or subscript depending on the context). These ℓ can be thought as $\ell \approx L/K$, it is the main asymptotic parameter in all the complexities, directly related to the number of elementary vector operations. In practice, for the key sizes and precision we consider above, all these ℓ's fall within the range $[1, 100]$

3 A Bivariate Polynomial Representation

We keep the notation and the setting from Section 2. Let K be a limb size. We represent (rational) approximations of elements of $\mathbb{R}[X]/\langle X^N + 1\rangle$ by elements of the (discrete) quotient ring $\mathbb{Z}[X, Y]/\langle X^N + 1\rangle$ (also equal to $\mathcal{R}[Y]$, the polynomials in Y over $\mathcal{R}$) of the bivariate polynomial ring $\mathbb{Z}[X, Y]$. The representations are obtained via the evaluation map (ring homomorphism) on the Y variable

$$\phi_K \colon \mathcal{R}[Y] \to \mathbb{R}_N[X], \qquad P(X, Y) \mapsto P(X, 2^{-K}). \tag{4}$$

More explicitly, the elements of $\mathcal{R}[Y]$ are represented by bivariate integer polynomials $P(X, Y) = \sum a_{i,j} X^i Y^j$ whose degrees in X are at most $N - 1$. Since the ring homomorphism ϕ_K is clearly not injective, an element of $\mathbb{R}_N[X]$ can have multiple pre-images in $\mathcal{R}[Y]$. We will use this property in a crucial way and will be particularly interested in representatives with small coefficients.

Definition 3.1. *A bivariate polynomial* $P(X, Y) = \sum_{i=0}^{N-1} \sum_{j \geq 0} a_{i,j} X^i Y^j$ *is called* K-normalized *if* $a_{i,j} \in [-2^{K-1}, 2^{K-1})$ *for all* $i = 0, \ldots, N - 1$ *and* $j \geq 1$.

As we are mainly interested in representing elements of $\mathbb{T}_N[X]$, that is, formal polynomials over the torus, we often reduce the coefficients of real polynomials to the real interval $[-1/2, 1/2)$ and hence, we say that

Definition 3.2. *A bivariate polynomial* $P(X,Y) = \sum_{i=0}^{N-1} \sum_{j \geq 0} a_{i,j} X^i Y^j$ *is* K-normalized *and* reduced *if, in addition to being* K*-normalized, it satisfies* $a_{i,0} = 0$ *for all* $i = 0, \ldots, N-1$*.*

We now approximate any element of the $\mathcal{R}$-module $\mathbb{T}_N[X]$ up to an arbitrary precision with K-normalized and reduced bivariate polynomials from $\mathcal{R}[Y]$. The proofs of the lemma and the corollary below are simple consequences of the decomposition of the coefficients in base 2^K.

Lemma 3.1. *For every integer* $L > 0$ *and every polynomial* $Q \in \mathbb{R}_N[X]$*, there exists a* K*-normalized polynomial* $P(X,Y) \in \mathcal{R}[Y]$ *of degree* $\leq \lceil L/K \rceil$ *in* Y *such that* $\|\phi_K(P) - Q\|_\infty \leq 2^{-L}$*.*

Corollary 3.1. *For all elements* $Q \in \mathbb{T}_N[X]$*, there exists a normalized and reduced polynomial* $P \in \mathcal{R}[Y]$ *such that*

$$\|\phi_K(P) \bmod \mathcal{R} - Q\|_\infty \leq 2^{-L}.$$

Note that the above approximation of the elements $Q \in \mathbb{T}_N[X]$ via bivariate polynomials $P(X,Y) \in \mathcal{R}[Y]$ is reminiscent to a *gadget decomposition* in classical lattice-based cryptography. Intuitively, the presence of the redundant variable Y corresponds to representing a vector in higher-dimensional space with lower $\|.\|_\infty$-norm - a key technique necessary for the design of various FHE schemes (e.g., `GSW` [24] and `TFHE` [14]).

The following lemma whose proof is rather formal shows that for any limb size K and any polynomial in $P \in \mathcal{R}[Y]$, there is a unique K-normalized and reduced polynomial Q with the same image under ϕ_K.

Lemma 3.2. *For any limb size* K *and any polynomial* $P(X,Y) \in \mathcal{R}[Y]$*, there exists a unique normalized polynomial* $Q(X,Y)$ *of the same degrees in both* X *and* Y *such that* $\phi_K(P) = \phi_K(Q)$*. Additionally, there exists a unique* K*-normalized and reduced polynomial* $Q(X,Y)$ *such that* $\phi_K(P) = \phi_K(Q) \bmod \mathcal{R}$*.*

We are now interested in an efficient algorithm for normalization and reduction of polynomials.

Lemma 3.3. *Let* K *be a limb size and let* $L > 0$ *be a specified precision. Given a polynomial* $A(X,Y) \in \mathcal{R}[Y]$ *satisfying* $\|A\|_\infty < 2^M$ *for some* M*, Algorithm 1 outputs a* K*-normalized and reduced polynomial* $R(X,Y) \in \mathcal{R}[Y]$ *such that* $\|(\phi_K(A) - \phi_K(R)) \bmod \mathcal{R}\| \leq 2^{-L}$ *in* $O((L+M)/K)$ *element-wise operations (additions/subtractions or binary shifts) on* N*-length vectors of integers in* $(-2^{M+1}, 2^{M+1})$*.*

A proof of this lemma is given in appendix Section A. Using a parameter value M such that $M+1 < 64$, Algorithm 1 allows for efficient `AVX` and GPU-friendly implementations.

Algorithm 1 Normalization and Reduction

```
Input: A target precision of $L$ bits (that represents $2^{-L}$) and a limb size $K$
Input: An input polynomial $A(X,Y) = \sum_{k\geq 0} A_k(X)Y^k$ satisfying $\|A\|_\infty < 2^M$
Output: A $K$-normalized and reduced polynomial $R(X,Y) = \sum_{k=1}^{\ell} R_k(X)Y^k$ of degree $\ell = \lceil L/K \rceil$ in $Y$ such that $\|\phi_K(R) - \phi_K(A)\|_\infty \leq 2^{-L}$.
1: $\mathtt{acc}(X) = 0$
2: for $k = \lceil (L+M)/K \rceil$ downto $1$ do
3:     $\mathtt{acc}(X) \leftarrow \mathtt{acc}(X) + A_k(X)$
4:     $R_k(X) \leftarrow \mathtt{centermod}_{2^K}(\mathtt{acc}(X))$
5:     $\mathtt{acc}(X) \leftarrow (\mathtt{acc}(X) - R_k(X))/2^K$
6: end for
7: Return $R(X,Y) = \sum_{k=1}^{\ell} R_k(X)Y^k$
```

Since the representations that occur throughout a FHE computation can be parameterized with different limb sizes, we are going to present the general theorems using two limb sizes K and $\widetilde{K}$ that are not necessarily the same. This is why in appendix Section B we present a slightly more general algorithm that converts a K-representation into a normalized $\widetilde{K}$-representation. Even if slightly more complex than Algorithm 1, it has the same performance.

3.1 Evaluation of external products

Most homomorphic operations reduce to efficient evaluations of Lipschitz functions. Recall that a function $f\colon \mathbb{T}_N[X] \to \mathbb{T}_N[X]$ is called κ-Lipschitz for some parameter $\kappa > 0$ if $\|f(x) - f(y)\|_\infty \leq \kappa \|x - y\|_\infty$ for all $x, t \in \mathbb{T}_N[X]$. Most of the time, the Lipschitz functions used in homomorphic evaluation are, in addition, $\mathcal{R}$-module homomorphisms. Since $\mathrm{Hom}_{\mathcal{R}}(\mathbb{T}_N[X], \mathbb{T}_N[X]) \simeq \mathcal{R}$, all Lipschitz functions that are $\mathcal{R}$-module homomorphisms are external products by a fixed element of $\mathcal{R}$, that is, $f_u\colon \mathbb{T}_N[X] \to \mathbb{T}_N[X]$, $x \mapsto u \cdot x$. We thus explain how to evaluate efficiently external products. The lemma below formalizes the following simple fact: evaluating f_u up to a certain target precision 2^{-L} amounts to evaluating the reduced and K-normalized representations of the external products $u \cdot 2^{-K}, \ldots, u \cdot 2^{-\widetilde{K}\widetilde{\ell}}$ for sufficiently large $\widetilde{\ell}$.

The following technical lemma enables us to compute a sufficiently good K-normalized and reduced representation of the external product $u \cdot \bullet$ with a prescribed target precision of L bits by providing (in a precomputation) sufficiently good approximations of the external product of u with the negative powers 2^{-Kj}.

Lemma 3.4. *Let $u \in \mathcal{R}$ be an integer polynomial and consider a target precision of $L > 0$ bits. Let K and $\widetilde{K}$ be two positive integers (limb sizes). Let $B_1(X,Y), \ldots, B_{\widetilde{\ell}}(X,Y)$ be K-normalized and reduced representations of $u \cdot 2^{-\widetilde{K}}, u \cdot 2^{-2\widetilde{K}}, \ldots, u \cdot 2^{-\widetilde{K}\widetilde{\ell}}$ with precision $(\widetilde{\ell}N)^{-1}2^{-(\widetilde{K}+L-1)}$. For any $\widetilde{K}$-normalized and* *reduced bivariate polynomial* $A(X,Y) = \sum_{i=1}^{\ell} A_i(X)Y^i \in \mathcal{R}[Y]$,

$$C(X,Y) = \sum_{i=1}^{\widetilde{\ell}} A_i(X)B_i(X,Y) \tag{5}$$

is a (non-reduced) K-representation of $u \cdot \phi_{\widetilde{K}}(A)$ of precision 2^{-L}, i.e.,

$$\left\|\phi_K(C) - u \cdot \phi_{\widetilde{K}}(A))\right\|_\infty \le 2^{-L} \qquad \textit{and} \qquad \|C\|_\infty \le \tilde{\ell}N2^{K+\widetilde{K}-2}.$$

Proof. The second bound on $\|C\|$ comes from the fact that A and B are resp K and $\widetilde{K}$-normalized, thus each $\|A_i\|_\infty$ and $\|B_i\|_\infty$ are bounded by resp. 2^{K-1} and $2^{\widetilde{K}-1}$. $\phi_K(C) = \sum_{i=1}^{\tilde{\ell}} A_i\phi_K(B_i) = \sum_{i=1}^{\tilde{\ell}} A_i(u \cdot 2^{-\tilde{K}i} + e_i)$ where by definition, each $\|e_i\|_\infty \le (\tilde{\ell}N)^{-1}2^{-(K+L-1)}$. Therefore, $\phi_K(C) = u \cdot \phi_{\widetilde{K}}(A) + \sum_{i=1}^{\tilde{\ell}} A_ie_i$, so the first inequality becomes $\left\|\sum_{i=1}^{\tilde{\ell}} A_ie_i\right\|$ which is $\le \sum_{i=1}^{\tilde{\ell}} N\|A_i\|_\infty \|e_i\|_\infty \le 2^{-L}$. □

If we expand further the right-hand side of (5) via $B_i(X,Y) = \sum_{j=1}^{\ell} B_{i,j}(X)Y^j$ for $\ell \le L + 1 + \tilde{K} + \log_2 \widetilde{\ell}$, we obtain $C(X,Y) = \sum_{j=1}^{\ell}\sum_{i=1}^{\tilde{\ell}} A_i(X)B_{i,j}(X)Y^j$ that can be computed using

- $\ell\tilde{\ell}$ internal polynomial products over $\mathcal{R}$ with inputs of infinity norm bounded by $2^{\tilde{K}-1}$ and 2^{K-1}, respectively and output of norm bounded by $N \cdot 2^{\tilde{K}+K-2}$.
- $\ell\tilde{\ell}$ additions of polynomials whose norm is bounded by $N\tilde{\ell} \cdot 2^{\tilde{K}+K-2}$.

In most FHE operations with GSW ciphertexts such as key switching, relinearization, automorphisms and bootstrapping, $u \in \mathcal{R}$ depends only on the secret key and is thus known in advance of the homomorphic operation. Anything that depends only on u can thus be precomputed in an offline phase (in general during the key generation) and only the cross-terms must be evaluated in an online phase (HE evaluation). With this in mind, should we decide to use DFT over only X, all the multiplications between $A_i(X)$ and $B_{i,j}(X)$ become element-wise products on the DFT space and are performed separately for each power of Y. Therefore, we obtain a nice offline/online phase separation:

Offline Phase (most often during keygen)
- $\ell\tilde{\ell}$ bounded DFT's of $B_{i,j}(X)$ precomputed and given as input

Online Phase
- $\tilde{\ell}$ bounded DFT's of $A_i(X)$
- $\ell\tilde{\ell}$ element-wise multiplications in DFT domain
- $\ell\tilde{\ell}$ element-wise additions in DFT domain
- ℓ bounded DFT's for the results $C_j(X)$
- $O(\ell)$ element-wise additions, shifts or masks to normalize the result (if needed)

Even if we have ℓ^2 products to evaluate, the online phase requires only a linear number of DFT's instead of a quadratic number. Fundamentally, it is the exact same root cause that lead [26] to its new asymptotic speedup compared to full-RNS. The main advantage here is that we obtain the representation in a natural way from the normalized gadget decomposition on $\mathbb{T}_N[X]$.

3.2 External products by secret polynomials over `TRLWE`

`TRGSW` ciphertexts have traditionally been used to multiply `TRLWE` ciphertexts by a secret integer polynomial $u \in \mathcal{R}$, which is the `TRGSW-TRLWE` external product from [15]. As noted in [6, p.326], the homomorphic evaluation of the external product $v \to s \cdot v$ where s is the small `TRLWE` secret key could be used as an alternative to the traditional relinearization of the quadratic term s^2 originally used in the `CKKS` and `BFV` products. This is done via a relinearization key $\mathtt{RK} = \mathtt{TRGSW}_s(s)$ using an external product of the form $\mathtt{RK} \boxdot (a, 0)$ (see [6, Defn.3]). However, once we propagate the zeros in this formula, we realize that the latter requires only half of a `TRGSW` ciphertext and half of the running time. Also $(a, 0)$ has to be treated as a noiseless ciphertext, which seems a bit arbitrary. These points are better formally explained below via a concept that we call `HalfRGSW` ciphertext[4]. The latter can be used to multiply a secret $u \in \mathcal{R}$ with a public $v \in \mathbb{T}_N[X]$ via the ⊾ secret-public external product operation:

$$\mathtt{HalfRGSW}(u) \mathbin{⊾} v \to \mathtt{TRLWE}(u \cdot v)$$

rather than via the $\boxdot$ and the full `TRGSW` ciphertext of s^2.

For a secret $u \in \mathcal{R}$ and ℓ `TRLWE` ciphertexts of zero $Z := (A|B) \in \mathbb{T}_N[X]^{\ell \times 2}$, the `HalfRGSW` is defined as

$$\mathtt{HalfRGSW}(u) := \big(A, B + u(B_g^{-1}, \ldots, B_g^{-\ell})\big) \in \mathbb{T}_N[X]^{2 \times \ell},$$

Note that for all valid `TRLWE` encryption $(a, b) \in \mathbb{T}_N[X]^2$ of v the element $\mathtt{HalfRGSW}(u) \mathbin{⊾} b - \mathtt{HalfRGSW}(us) \mathbin{⊾} a \in \mathbb{T}_N[X]^2$ is a valid `TRLWE` ciphertext (under s) of the plaintext $u \cdot v$. That yields a more flexible computation of a ciphertext of $u \cdot v$ instead of computing the traditional external product $\mathtt{TRGSW}(u) \boxdot (a, b)$, thus, justifying the principle that *two halves make a whole.* As a bonus, a speed-up can be achieved by using different parameters for the two halves, especially in small levels as in the bootstrapping of `TFHE`.

Definition 3.3 (bivariate `RLWE` ciphertexts). *Let K be a limb size, a `bivRLWE` ciphertext C under a small key $S \in \mathbb{Z}_N[X]$ of the message $m \in \mathbb{T}_N[X]$ is materialized by a tuple $(A, B) \in \mathcal{R}[Y]^2$ of K-normalized and reduced representations whose phase $\varphi_{S,K}(A, B) \underset{def}{=} \phi_K(B) - S \cdot \phi_K(A)$ is equal to $m + e$ where $e \in \mathbb{T}_N[X]$ is a small Gaussian error. We will note $\mathtt{bivRLWE}_{S,K,2^{-L}}(m)$ such bivariate `RLWE` encryption of m with error bound 2^{-L}*

Note that `bivRLWE` ciphertexts satisfy the following *prefix property*: a higher precision ciphertext with small error yields a lower precision one by simply restricting the representation to the first few limbs (prefix). For instance, when restricting a high precision ciphertext (A, B) of error norm $\leq 2^{-L+1}$ to degrees $\ell_B = \lceil (L+2)/K \rceil$ and $\ell_A = \lceil (L+2+\log_2 \|S\|_1)/K \rceil$ in B and A, respectively, we

[4] We could also name this ciphertext $RK(u)$ since it has the shape of a relinearization key. However the GSW name better depicts the fact that it is a ciphertext not some key material, and most importantly, the ⊾ morphism is half of the $\boxdot$ operation.

obtain an encryption of the same plaintext with error $\leq 2^{-L}$. The importance of the *prefix property* is that it is a computation-free version of modulus switching. We assume throughout that all `bivRLWE` ciphertexts of precision 2^{-L} are instantiated with degrees ℓ_A, ℓ_B in Y. We also consider the bivariate `HalfRGSW` counterpart:

Definition 3.4 (bivariate `HalfRGSW` encryptions). *Let $K, \widetilde{K}$ be limb sizes, let $u \in \mathcal{R}$ be a small polynomial of norm $\|u\|_1 \leq \kappa$ and let $S \in \mathcal{R}$ be a small key. We define $\texttt{bivHalfRGSW}_{S,K,\tilde{K},2^{-L}}(u)$ (bivariate half `RingGSW` encryption of u under S with precision L) to be a family of $\texttt{bivRLWE}_{S,K,2^{-L}}$ ciphertexts of $u \cdot 2^{-\tilde{K}}, u \cdot 2^{-2\tilde{K}}, \dots, u \cdot 2^{-\tilde{\ell}\tilde{K}}$. The family can be restricted to its first $\tilde{\ell} = \lceil (L + 2 + \log_2 \kappa)/\widetilde{K} \rceil$ ciphertexts. If the `bivHalfRGSW` encryption is given in `DFT` basis, we denote it by $\texttt{bivHalfRGSW}^{\texttt{DFT}}$.*

These ciphertexts also satisfy an even stronger prefix property: from any `bivHalfRGSW` ciphertext C of high precision $\leq 2^{-(L+1)}$, the truncations to degrees ℓ_A, ℓ_B above of its first $\tilde{\ell} = \lceil (L + 2 + \log_2(\kappa))/\widetilde{K} \rceil$ `bivRLWE` ciphertexts form a `bivHalfRGSW` of the same message with lower precision $\leq 2^{-L}$.

Because of the prefix property, these ciphertexts can be passed to any function that require a lower precision level $L' < L$: in this case, the function will only access the terms of degree $\ell'_A \leq \ell_A$ and $\ell'_B \leq \ell_B$ from the first $\tilde{\ell}' \leq \tilde{\ell}$ elements. This property holds both on paper and also in efficient implementations, where ciphertexts are passed by pointers.

Theorem 3.1 (half external product (secret×public)). *Let $u \in \mathcal{R}$ be a small polynomial of norm $\|u\|_1 \leq \kappa$ for some $\kappa > 0$, let $a \in \mathbb{T}_N[X]$ and let L an output precision parameter. Let $L_\alpha, L_\beta \geq L$ be parameters satisfying $2^{-L_\alpha} + 2^{-L_\beta} \leq 2^{-L}$ and let $L_1 = L_\alpha + \log_2 \kappa$, $\tilde{\ell} = \lceil L_1/K \rceil$ and $L_2 = L_\beta + \widetilde{K} + \log_2(N\tilde{\ell})$. If $C_f = (c_1, \dots, c_{\tilde{\ell}})$ is a $\texttt{bivHalfRGSW}^{DFT}_{S,K,\tilde{K},2^{-L_2}}$ encryption of u with precision $\leq 2^{-L_2}$ and $A(X,Y) = \sum_{i=1}^{\tilde{\ell}} A_i \cdot Y^i$ is a $\widetilde{K}$-normalized and reduced representation of $a \in \mathbb{T}_N[X]$ up to 2^{-L_1} then the ciphertext*

$$C_f \boxdot a = \texttt{normalizeReduce}\left(\texttt{iDFT}\left(\sum_{i=1}^{\tilde{\ell}} \texttt{DFT}(A_i) \cdot c_i\right)\right)$$

is a $\texttt{bivRLWE}_{S,K,2^{-L}}$ encryption of $u \cdot a$. Homomorphic evaluation of such an external product with $\tilde{\ell} = \lceil L_1/\tilde{K} \rceil$ and $\ell = \lceil (L_2 + \log_2 N + 2)/K \rceil$ requires

- *$\tilde{\ell}$ bounded DFT's of the A_i's of norm $\leq 2^{\tilde{K}-1}$,*
- *$2\ell\tilde{\ell}$ element-wise `addmul`'s in DFT domain for polynomials of norm $\leq N\tilde{\ell}2^{\tilde{K}+K-2}$,*
- *2ℓ bounded DFT's for the results $C_j(X)$ with norm bound $N\tilde{\ell}2^{\tilde{K}+K-2}$,*
- *2ℓ element-wise additions/shifts/masks to normalize and truncate the result.*

Proof. Letting $e = a - \sum_{i=1}^{\tilde{\ell}} A_i \cdot 2^{\tilde{K}i}$, we have $\|e\|_\infty \leq 2^{-L_1}$. If $c_i' = \texttt{iDFT}(c_i)$ for $i = 1, \dots, \tilde{\ell}$ then $\varphi_{S,K}(c_i') = u \cdot 2^{-\tilde{K}i} + e_i$ where $\|e_i\|_\infty \leq 2^{-L_2}$. Since $C_f \boxdot a =$

$\texttt{normalizeRed}(\sum_{i=1}^{\tilde{\ell}} A_i.c_i')$, it follows that $\varphi_{S,K}(C_f \boxslash a) = \sum_{i=1}^{\tilde{\ell}} A_i.\varphi_{S,K}(c_i') = u \cdot \left(\sum_{i=1}^{\tilde{\ell}} A_i \cdot 2^{-\tilde{K}i}\right) + \sum_{i=1}^{\tilde{\ell}} A_i \cdot e_i$, and therefore, $\|\varphi_{S,K}(C_f \boxslash a) - u \cdot a\|_\infty \leq \|u \cdot e\|_\infty + \sum_{i=1}^{\tilde{\ell}} \|A_i \cdot e_i\|_\infty \leq 2^{-L_\alpha} + 2^{-L_\beta} \leq 2^{-L}$. The rest of the theorem is a simple count of operations, the degrees of c_i being bounded by ℓ. $\square$

The above theorem applies to the traditional key switching, automorphisms and relinearization operations. To recover the traditional homomorphic product of a secret u by a `bivRLWE` encrypted ciphertext $(a, b) \in \mathcal{R}[Y]^2$ under a small key $s \in \mathbb{Z}_N[X]$, we can use one `bivHalfRGSW` ciphertext C_u of u and one `bivHalfRGSW` ciphertext C_{su} of su, and call the pair $C = (C_u, C_{su})$ a full `bivRGSW`(u). The full external product is then

$$C \boxdot (a, b) = C_u \boxslash b - C_{su} \boxslash a. \tag{6}$$

Note that since the norms $\|u\|_1$ and $\|su\|_1$ in the two halves are distinct, Theorem 3.1 suggests the use of distinct parameters for each half. This reflects the natural property that in a RingLWE ciphertext (a, b), it suffices to provide the term b up to a lower precision compared to the term a. In the next section, we show that this speeds up the bootstrapping of `TFHE` by removing 12.5% of the FFTs and 16.7% of the products and decompositions.

We also merge the final normalizations of the two half-products together to obtain the following

Corollary 3.2 (full external product (secret×secret)). *Let $K, \widetilde{K}, \widetilde{K}'$ be limb sizes, let $u \in \mathcal{R}$ be a small polynomial, let $v \in \mathbb{T}_N[X]$ be a message, let $s \in \mathcal{R}$ a small key and and $2^{-L} > 0$ be a target output precision. For all $L_\alpha, L_\beta, L_\gamma \geq L$ satisfying $2^{-L_\alpha}+2^{-L_\beta}+2^{-L_\gamma} \leq 2^{-L}$, let $L_1 = L_\alpha+\log_2 \|u\|_1$, $\tilde{\ell}_B = \lceil (L_1+2)/\widetilde{K} \rceil$, $\tilde{\ell}_A = \lceil (L_1 + 2 + \log_2 \|s\|_1)/\widetilde{K}' \rceil$, $L_2 = L_\beta + \widetilde{K} + \log_2(N\tilde{\ell}_B)$ and $L_2' = L_\gamma + \widetilde{K}' + \log_2(N\tilde{\ell}_A)$. If $(A, B) \in \mathcal{R}[Y]^2$ is a $\texttt{bivRLWE}_{s,K}$ of $v \in \mathbb{T}_N[X]$ with noise $\leq 2^{-L_1}$, if $C_u = (c_1, \ldots, c_{\tilde{\ell}})$ is a $\texttt{bivHalfRGSW}_{s,K,\widetilde{K}}$ encryption of u with precision 2^{-L_2} and if $C_{su} = (d_1, \ldots, d_{\tilde{\ell}})$ is a $\texttt{bivHalfRGSW}_{s,\widetilde{K}'}$ encryption of su with precision $2^{-L_2'}$ then*

$$(C_u, C_{su}) \boxdot (a, b) = \texttt{normalizeRed}\left(\texttt{iDFT}\left(\sum_{i=1}^{\tilde{\ell}_B} \texttt{DFT}(B_i) \cdot c_i - \sum_{i=1}^{\tilde{\ell}_A} \texttt{DFT}(A_i) \cdot d_i\right)\right),$$

is a $\texttt{bivRLWE}_{S,K,2^{-L}}$ encryption of $u \cdot v$. Here, A and B have been $\widetilde{K}'$ and $\widetilde{K}$-normalized, respectively. In addition, computing this encryption with $\ell = \lceil (L_2' + \log_2 N + 2)/K \rceil$ requires at most

- *$\tilde{\ell}_A + \tilde{\ell}_B$ bounded DFT's of the A_i, B_i's of norm $\leq 2^{\widetilde{K}-1}$,*
- *$2\ell(\tilde{\ell}_A + \tilde{\ell}_B)$ element-wise `addmul`'s in DFT domain for polynomials of norm $\leq 2N\tilde{\ell}2^{\widetilde{K}+K-2}$,*
- *2ℓ bounded DFT's for the results $C_j(X)$ with norm bound $2N\tilde{\ell}2^{\widetilde{K}+K-2}$,*

– *$\widetilde{\ell}_A + \widetilde{\ell}_B + 2\ell$ element-wise additions/shifts or masks to normalize and truncate the input ciphertext and the final result.*

Proof. Let $d'_i = \texttt{iDFT}(d_i)$ and $c'_i = \texttt{iDFT}(c_i)$, by the same proof as in Theorem 3.1, $\varphi_{S,K}(\sum_{i=1}^{\widetilde{\ell}_B} B_i \cdot c'_i) = u \cdot \phi_{\widetilde{K}}(B) + e_1$ where $\|e_1\|_\infty \leq 2^{-L_\beta}$ and $\varphi_{S,K}(\sum_{i=1}^{\widetilde{\ell}_A} A_i \cdot d'_i) = su \cdot \phi_{\widetilde{K}'}(A) + e_2$ where $\|e_2\|_\infty \leq 2^{-L_\gamma}$. Therefore, $\varphi_{S,K}(C \boxdot (A,B))$ is the difference $u \cdot (\phi_{\widetilde{K}}(B) - s \cdot \phi_{\widetilde{K}}(A)) + e_1 - e_2 = u \cdot \varphi_{S,K}(A,B) + e_1 - e_2$. Since $\varphi_{S,K}(A,B) = v + e$ where $e \leq 2^{-L_1}$, we have $\|\varphi_{S,K}(C \boxdot (A,B)) - u \cdot v\|_\infty \leq 2^{-L_\alpha} + 2^{-L_\beta} + 2^{-L_\gamma} \leq 2^{-L}$. □

3.3 Public linear combinations

The main difference and advantage of the bivariate representation, over the more classical base-2^K representation is the ability to decouple and delay carry propagation, which leaves the opportunity to do a lot of linear algebra between two normalizations. The bivariate representation is linear over $\mathbb{Z}$ and $\mathbb{Z}_N[X]$, so linear combinations $\sum_{i=1}^{k}(\lambda_i.m_i)$ of ciphertexts can primarily be evaluated termwise. From a normalized representation, if the intent of normalization is to maintain the base arithmetic bounded, e.g. by $O(\ell.N.2^{2K})$ like in the external product, it leaves enough room to evaluate more than $\ell.N.2^K$ (so more than 100000) simple additions and subtractions before a single normalization is needed. To evaluate linear combinations with larger coefficients, where $\|\lambda_i\|_\infty$ are larger than 2^K, we can use the same strategy as for the external product: decompose the integer coefficients λ_i in base 2^K to represent them under the form $\sum_{j=1}^{p} e_j 2^{Kj}$, and precompute the DFT of the e_j's. ($p = 2$ or 3 terms are sufficient to represent the constant coefficients λ_i that appear in a BFV or CKKS bootstrapping). The linear combination is then easy to apply under this form, and requires only $O(p\ell)$ element-wise products, which is still negligible compared to the cost of an external product.

3.4 Automorphisms in `BFV` and `CKKS`

Unlike external products that can operate with very short keys and large noise, `BFV` and `CKKS` arithmetic usually operate on much larger parameters, where a single limb is in general the optimal choice. For the rest of the section, we will therefore consider that there is a unique limb size K (i.e. $K = \widetilde{K}$).

Automorphisms of $\mathbb{T}_N[X]$ are $\mathcal{R}$-module homomorphisms. These are the *rotation/conjugation* functions $\sigma_k \colon \mathbb{T}_N[X] \to \mathbb{T}_N[X]$ that substitutes the variable X with X^k where k is odd. $\sigma_k(a)$ can be computed efficiently in coefficient space over the bivariate representations $\mathbb{Z}[X,Y]/\langle X^N + 1\rangle$ by treating each power of Y independently and mapping $\sum_{i=1}^{\ell} A_i(X)Y^i$ to $\sum_{i=1}^{\ell} \sigma_k(A_i(X))Y^i$.

Over ciphertexts, we just need to observe that $\sigma(b - sa) = \sigma(b) - \sigma(s) \cdot \sigma(a)$, so given an `bivHalfRGSW` encryption of $\sigma(s)$, we recover the well known homomorphic evaluation:

Lemma 3.5 (Automorphism in BFV and CKKS). *Let K be a limb size, let $v \in \mathbb{T}_N[X]$ be a message, let σ be an automorphism of $\mathbb{T}_N[X]$ and let L be an output precision parameter. For all $L_\alpha, L_\beta \geq L$ that satisfy $2^{-L_\alpha} + 2^{-L_\beta} \leq 2^{-L}$, if $(a, b) \in \mathcal{R}[Y]^2$ is a $\texttt{bivRLWE}_{s,K}$ encryption of $v \in \mathbb{T}_N[X]$ with noise 2^{-L_α} and $C_{\sigma(s)}$ is a $\texttt{bivHalfRGSW}_{s,K,K}$ of $\sigma(s)$ with noise 2^{-L_β}, then*

$$\boxed{\sigma}((a, b)) = (0, \sigma(b)) - C_{\sigma(s)} \triangleright \sigma(a)$$

is a $\texttt{bivRLWE}_{s,K}$ encryption of $\sigma(v)$ with noise $\leq 2^{-L}$. Note that we can omit the `normalizeReduce` *operation of the $\triangleright$ operation in Theorem 3.1 and compute it at the end, after the subtraction. Computing this operation with $\widetilde{\ell} = \lceil L_\alpha/K \rceil$ and $\ell = \lceil (L_\beta + \log_2 N + 2)/K \rceil$ requires, as in Theorem 3.1,*

- *$\widetilde{\ell}$ bounded DFT's of the A_i's, of norm $\leq 2^{K-1}$,*
- *$2\ell\widetilde{\ell}$ element-wise* `addmul`*'s in DFT domain for polynomials of norm $\leq N\widetilde{\ell}2^{2K-2}$,*
- *2ℓ bounded DFT's for the results $C_{\sigma(s)}(X)$ with norm bound $N\widetilde{\ell}2^{2K-2}$,*
- *2ℓ element-wise additions/shifts/masks to normalize and truncate the final result,*
- *(in addition to the computations in Theorem 3.1), 2ℓ evaluations of σ over the input A_i and B_i, of norm bound 2^K.*

Proof. We first verify that $\varphi_{s,K}(\boxed{\sigma}((a, b)))$ is close to $\sigma(v)$. We compute the noise level of the output as $\|\varphi_{s,K}(\boxed{\sigma}((a, b)) - \sigma(v))\|_\infty = \|\phi_K(\sigma(b)) - \varphi_{s,K}(C_{\sigma(s)} \triangleright \sigma(a)) - \sigma(v))\|_\infty = \|\sigma(\phi_K(b))) - \sigma(v) - \sigma(s\phi_k(a)) + \sigma(s\phi_k(a)) - \varphi_{s,K}(C \triangleright \sigma(a))\|_\infty \leq \|\sigma(\phi_K(b) - s\phi_K(a) - v)\|_\infty + \left\|\sigma(s\phi_K(a)) - \varphi_{s,K}(C_{\sigma(s)} \triangleright \sigma(a))\right\|_\infty$. Per the LWE definition, and because σ is an isometry, $\|\sigma(\phi_K(b) - s\phi_K(a) - v)\|_\infty = \|\sigma(e)\|_\infty = \|e\|_\infty \leq 2^{-L_\alpha}$. Per Theorem 3.1, the second term is a (half) external product noise bounded by $\left\|\sigma(s)\sigma(\phi_K(a)) - \varphi_{s,K}(C_{\sigma(s)} \triangleright \sigma(a))\right\|_\infty \leq 2^{-L_\beta}$. Summing the two, the output noise is bounded by $2^{-L_\alpha} + 2^{-L_\beta} \leq 2^{-L}$.

□

3.5 Internal products in BFV and CKKS

In the previous section, we showed how to efficiently compute homomorphic external products and automorphisms using leveled-FFT. We now discuss how to efficiently compute homomorphic internal products between two `TRLWE` ciphertexts as defined in Section 2 and in [6].

Using the notation of Section 2, to evaluate a `BFV` or `CKKS` product between two `RingLWE` ciphertexts, we first apply (6) to the relinearization term of (1):

$$\texttt{TRGSW}(s) \boxdot (P_2, 0) = \texttt{bivHalfRGSW}(s) \triangleright 0 - \texttt{bivHalfRGSW}(s^2) \triangleright P_2$$

After substituting the expression in (1), the internal product becomes

$$(a_1, b_1) \boxtimes (a_2, b_2) = (P_1, P_0) + \texttt{bivHalfRGSW}(s^2) \triangleright P_2, \tag{7}$$

where $P_0 = b_1 \otimes_\star b_2$, $P_2 = a_1 \otimes_\star a_2$, $P_1 = a_1 \otimes_\star b_2 + b_1 \otimes_\star a_2$ for $\star = \{\texttt{BFV}, \texttt{CKKS}\}$ depending on whether one computes the `BFV` product (2) or the `CKKS` product (3).

We compute the internal product using that formula by first approximating the inputs by bivariate polynomials via the evaluation ring homomorphism (4). Lifts from the torus to the real fields come for free, and rounding in CKKS is a simple truncation and bit masking operation on the last limb. We then use DFT bivariate polynomial multiplication (in X and Y) to evaluate the products $\otimes_\star$ and hence, compute approximations of P_0, P_1 and P_2. We use DFT over X and Y for the latter which runs in asymptotic complexity $O(\ell \log_2 \ell)$ SIMD operations on vectors of size N. This computation is asymptotically negligible compared to the half `TRGSW` product ⊾ in (7) that requires $2\ell^2$ such SIMD operations. In practice, even for smaller dimensions where one may use polynomial multiplication that is quadratic in ℓ, the run-time of computing $\otimes_\star$ never exceeds 75% of the run-time of computing ⊾. We summarize the complexity in the theorem below and provide detailed discussion of the optimizations in appendix Section C.

Theorem 3.2 (`CKKS/BFV` product complexity). *The homomorphic `CKKS` or `BFV` product between two* `bivRLWE` *ciphertexts* (a_1, b_1) *and* $(a_2, b_2) \in \mathcal{R}[Y]^2$ *with a relinearization key* $RK = \texttt{bivHalfRGSW}(s^2)$ *where* s *is the secret key requires:*

- 8ℓ *DFT/IDFTs in* X *(including* 3ℓ *of them for the external product)*
- $2\ell^2 + \min\left(7\ell \log_2 \ell + 9\ell, \frac{3}{2}\ell(\ell-1) + 4\ell\right)$ *SIMD operations (add, mul, addmul or twiddle factors) on vectors of size* N *(including* $2\ell^2$ *for the external product).*
- 3ℓ *SIMD rounding/normalization operations on vectors of size* N *(including* 2ℓ *of them in the external product).*

This lemma shows that as ℓ grows, the asymptotic complexity of an internal product is exactly the same as that of the underlying external product (i.e. $O(\ell^2 N)$) and the overhead induced by the rest of the operations is negligible. In practice, since the SIMD operations dominate, the run-time is at most 1.75 times the running time of the underlying external product for small values of ℓ.

We conclude this section with Table 1, which recalls the number of N-dimensional DFT's and the number of SIMD operations over vectors of $N/2$ complexes, for one half external product (so either one keyswitch or one automorphism), as well as the number of such operations in one `CKKS` or `BFV` product. Although the maximal levels for L for 128-bit security are 1761 and 880 for $N = 65536$ and 32768 respectively, we use $L = 1729$ and $L = 865$ as in the implementation provided by the authors of [26] to facilitate the comparison with their work. The table confirms that the larger we can choose the limb size K, the less elementary operations are required. We will show in the following sections that each choice of backend naturally comes with one maximal value of K it can process: for instance, the double floating points FFT is limited to $K = 19$, but other 128-bit constructions can afford larger limb sizes, which we explain in Section 6 together with some benchmarks.

The dominant part of the computation is spent in the SIMD products (or assimilated), the cost of the DFTs remains very small. $K = 19$ is the largest limb-size achieved via `float64` FFT whereas $K = 52$ is achieved via 120-bit NTT. Because elementary operations are faster in the first case, it compensates the

Table 1. Number of operations in one half-external product (i.e. one relinearization, one keyswitch, or one automorphism) and in one `BFV/CKKS` internal product (IP).

	$N = 65536,\ L = 1729$					$N = 32768,\ L = 865$				
K	ℓ	# DFTs	# SIMD prods	# IP DFTs	# IP SIMD prods	ℓ	# DFTs	# SIMD prods	# IP DFTs	# IP SIMD prods
16	109	327	23762	763	30278	55	165	6050	385	8927
19	**91**	**273**	**16562**	**637**	**22889**	**46**	**138**	**4232**	**322**	**7016**
22	79	237	12482	553	18683	40	120	3200	280	5600
...										
46	38	114	2888	266	5054	19	57	722	133	1263
49	36	108	2592	252	4536	18	54	648	126	1134
52	**34**	**102**	**2312**	**238**	**4046**	**17**	**51**	**578**	**119**	**1011**

additional number of operations, and the two backends end up give similar final running time: SIMD products for $N = 65536$ are micro-benchmarked at $27\mu s$ in the `float64` scenario, whereas the equivalent counterpart for 120-bit NTT take $93\mu s$. The number of SIMD products per internal product in Table 1 is an upper-bound that considers the best choice between the naive multiplication in $3\ell(\ell-1)/2$ and the DFT one in $O(\ell \log_2(\ell))$. Because ℓ has to be rounded up to the next power of two, some nodes in the resulting Cooley-Tuckey recursion would compute zeros: these nodes have been eliminated from the count in this table.

4 Accelerating `TFHE` Gate Bootstrapping

In this section, we show how the concept of `halfTRGSW` can speed-up the `TFHE` library [2]. One `TFHE` gate bootstrapping is computing $n = 630$ successive `TRGSW-TRLWE` external products. We defer to the noise propagation analysis in [14] and the lattice estimator [3] for the explanations on lattice security but in summary, there are two constraints on the external products:

- *Lattice security constraint.* Any `TRLWE` or `TRGSW` ciphertext encrypted with an $N = 1024$ dimensional key (binary or ternary) must have a noise variance parameter of at least $V_{\text{GSW}} = 2^{-50}$ (stdev 2^{-25}) to provide 128-bits of security.
- *Correctness of bootstrapping constraint.* Each individual external product output shall make the noise variance grow by at most $\Delta_{\max} = 8.4961.10^{-8}$ to ensure that the final ciphertext is decryptable.

Since the ratio between $\Delta_{\max}/V_{\text{GSW}}$ is very small, we have to pick the external product parameters with extreme care. For instance, it is unrealistic to try to take $K = \widetilde{K}$, instead, the choice in `TFHE` is to decrease $\widetilde{K}$ as much as needed to contain the noise growth of `TRGSW` ciphertexts, and maintain K to a fixed value 32, so that `TRLWE` use a single limb of 32-bits, and correspond to our `bivRLWE` with $\ell = 1$. Also, the worst-case bounds on noise amplitude increase too fast compared to the reality: as it is shown in LWE, we can use the average-case noise propagation formula, which essentially transcriptions of the worst-case theorems where all noises are assimilated to independent Gaussian samples and the theorem operates on their variance instead of their infinity norm.

With this in mind, in `TFHE`, the noise propagation of a `TRGSW-TRLWE` external product (adapted to use the notations of this paper), between a `TRGSW` ciphertext of error variance V_{TRGSW} for a decomposition in $\tilde{\ell}$ limbs of $\widetilde{K}$ bits, and an input TRLWE ciphertext of noise variance V_{in}, is:

$$V_{\text{out}} - V_{\text{in}} \leq 2\tilde{\ell}N4^{\widetilde{K}-1}V_{\text{GSW}} + (1 + \underbrace{N}_{\|S\|_2^2})4^{-\widetilde{K}\tilde{\ell}-1}. \tag{8}$$

The choices of $\widetilde{K}, \tilde{\ell}$ in `TFHE` library are 7 and 3, which correspond to a variance growth of $2.2410.10^{-8} < \Delta_{\max}$ per external product. As we know that the running time of `TFHE` is proportional to the number of FFTs per external products, this number is $2\tilde{\ell} + 2 = 8$.

We can also see that any attempt to reduce the number of FFTs by setting $\tilde{\ell} = 2$ for instance fails, as no more limb size $\widetilde{K}$ exists that makes $V_{\text{out}} - V_{\text{in}}$ in Eq. (8) smaller than $\Delta_{\max}$. That's precisely where the concept of `HalfRGSW` helps: if the input `TRLWE` is (A, B), and we are allowed to pick different parameters and dimensions $\widetilde{K}_A, \tilde{\ell}_A$ and $\widetilde{K}_B, \tilde{\ell}_B$, the variance growth bound of Eq. (8) becomes, by analogy with our Corollary 3.2:

$$\left(\tilde{\ell}_A N 4^{\widetilde{K}_A-1}V_{\text{GSW}} + N \cdot 4^{-\widetilde{K}_A\tilde{\ell}_A-1}\right) + \left(\tilde{\ell}_B N 4^{\widetilde{K}_B-1}V_{\text{GSW}} + 1 \cdot 4^{-\widetilde{K}_B\tilde{\ell}_B-1}\right)$$

Not unsurprisingly, the term in $\widetilde{K}_A, \tilde{\ell}_A$ is already tight, so the value $(7, 3)$ remains optimal. However the absence of the factor N in the second term (analogue of $\|s\|_1$ in worst-case formulae, which only affects the half external product term in A) gives us much more flexibility on the choices of $\widetilde{K}_B, \tilde{\ell}_B$. It turns out that we can finally reduce $\tilde{\ell}_B$ to 2 and use $\widetilde{K}_B = 8$. We indeed obtain the following variance growth: $2.986053.10^{-8}$ for the half external product in A and $1.1234.10^{-8}$ for the one in B, and we verify that the sum $4.10946.10^{-8}$ stays $\leq \Delta_{\max}$, and is thus suitable for bootstrapping.

Because of that improvement, the number of FFTs in `TFHE` decreases to $\tilde{\ell}_A + \tilde{\ell}_B + 2 = 7$ instead of 8 per external product (so 12.5% less FFTs), and the other linear operations, decompositions, and matrix multiplications also drop by 16.6%, since the outer loop has only 5 iterations instead of 6.

We first implemented this concept as a simple patch to the original `TFHE` library by overriding the `tGswExternMulToTLwe` function and removing the last iteration of the loop. On a n2-standard Xeon, we already witnessed a decrease of the `NAND` gate bootstrapping by 3ms which matches the expected theoretical speedup. In order to compare our proof-of-concept to the fastest CPU bootstrapping whose reference implementation is the TFHE-rs library [38], we then extracted and re-implemented the entire `blindrotate_woKS` loop obtaining another

Table 2. Performance comparison of gate bootstrapping with a n2-standard GCP instance with 64GB of RAM and a 12-th Gen i7-1260p laptop with 64GB of RAM. All the benchmarks are single core.

Library	Instruction set	n2-standard	12-Gen i7-1260p
TFHE-lib, spqlios-fma	AVX2	22.4ms	10.4ms
TFHE-rs, `TFHE_LIB_PARAMETERS`	AVX2	18.2ms	8.6ms
	AVX512	14.4ms	*not supported*
TFHE-rs, `DEFAULT_PARAMETERS`	AVX2	14.4ms	7.6ms
	AVX512	13.7ms	*not supported*
Our work, halfTRGSW	AVX2	11.2ms	5.3ms

2x speedup by back-porting the following engineering improvements: from the `TFHE-rs` optimizations, we used the fast `AVX` floating-point flooring, conversions to and from integer and bit-decompositions that work on bounded floating-point numbers that are never infinite, NaN, or subnormal. We also dropped completely the round-trip to the `int32/int64` and reduced the numbers modulo $\mathbb{Z}$ directly over their floating point bits, so that the entire blind rotate procedure operates solely on the double-floating point precision domain. Finally, we incorporated the latest improvements from fast implementations of `Falcon` to the complex FFT described in [31, sec.5]: namely instead of evaluating the $\log_2(N/2)$ iterations of the FFT circuit one after the other, we execute them two by two and thus save half of the memory accesses. The last four iterations, that operate contiguously on vectors of 16 complex numbers are run entirely on registers. We did however not backport the `DEFAULT_PARAMETERS` optimization of `TFHE-rs` that consists of trading the ring dimension N against an increase of the module dimension, nor any `AVX512` optimizations. Our prototype uses therefore only `AVX2` instructions. Finally, since we did not have access to a Xeon Platinum, we have run our experiments on two different architectures: one "slow" cloud instance, which is a GCP n2-standard Xeon CPU Cascade Lake at 2.8GHz with 4 cores, 8vCPU and 64GB of RAM (left column of Table 2), and one "fast" laptop with a AlderLake Core i7-1260P at 4.7GHz with 64GB of RAM (right column of Table 2). The fast laptop does not have `AVX512`, whose support was discontinued by Intel, but it has a much higher cache and memory access rates than the Cascade Lake counterpart, so timings on this machine are often very close to those published with a bare metal server with a Xeon Platinum: overall, Table 2 gives a neat intuition of the range of performance we can expect depending on the CPU. The combination of the half external product and all the other engineering optimizations described in this paragraph make our new bootstrapping running time as low as 5.3 milliseconds per `NAND` gate, which is the new single core record, even though it sticks to a traditional ring-lwe parameter-set and does not use any `AVX512` instructions.

5 Frontend and Large Number Arithmetic: Bivariate versus CRT Representations

The main idea in both [26] and the present work is the decoupling of the cyclotomic arithmetic (the backend representation) from the large number arithmetic (the frontend representation) in `BFV`, `CKKS`, `TFHE` and `Chimera` which, in turn, allows to fully benefit from the small polynomial coefficients produced by the gadget decomposition. Taking one step further, both works suggest a frontend/backend separation where TRLWE ciphertexts are represented non-uniquely by their gadget decomposition: a vector of small polynomials $\in (\mathbb{Z}_N[X])^\ell$. The content of the vector being the coefficients in Y for the bivariate representation, the centered and reduced coefficients mod q_i for CRT representations, and in general any other gadget-decomposed representation. Similarly, half-TRGSW ciphertexts would correspond to a matrix of $\ell \times 2\ell$ small integer polynomials. The most complex operation that arises in the external product is an efficient vector $\times$ matrix product, where the matrix is preprocessed in an offline phase. The online phase of the product is carried out by the backend API using the appropriate DFTs. Cheaper operations are of course, element-wise additions, scaling, rotations and automorphisms, as well as normalization whose role is to keep the representation small. Depending on the frontend, the bivariate approach benefits from the fact that modulus rescaling just requires a prefix truncation of the normalized representation, but has a $O(\ell.\log_2(\ell))$ $\otimes_\star$ products. CRT frontends have a more expensive modulus rescaling, but faster $O(\ell)$ internal products. With the operations described in Section 3.2, the bivariate and CRT frontends can be instantiated on the same backend API and offer similar performance. CRT frontends should be preferred for use-cases involving large homomorphic matrix products, as the external products and modulus rescaling can in these cases be amortized, and the frontend benefits from the faster $\otimes_\star$ products. On the opposite, the bivariate frontend is preferred when the use cases have fewer internal products and rely on lookup-tables, trace algorithms, circuit bootstrappings or when the homomorphic internal products are sequential. These use cases benefit from the fact that $\otimes_{\texttt{CKKS}}$ can be instantiated at any noise level L on the bivariate frontend, rather than at integer multiplicative levels (multiples of $\log_2(q_i)$) in the CRT frontend. A more in-depth comparison is provided in appendix Section F. Finally, since the external product is fast on both frontends (the CRT one via [26], and the bivariate one from Section 3.2), it is easy to switch dynamically between both on more complex use cases, which is also interesting from a scheme switching perspective.

6 Backend Arithmetic and Cyclotomic Multiplications: Approximate FFT or NTT

At a low level, we need efficient arithmetic (additions and multiplications) in the cyclotomic ring $\mathcal{R}$ where the coefficients of the polynomials are integers bounded by $B'_{\texttt{worst}} = 2\ell N 2^{K+\widetilde{K}-2}$ in the worst case. We can get a tighter average case bound if we pay closer attention to the expressions present in the previous section. We then notice that it is sufficient for such arithmetic to be able to successfully evaluate, with overwhelming probability, expressions of the form $\sum_{i=1}^{2\ell} a_i(X) \cdot b_i(X)$ where a, b have their coefficients computationally indistinguishable from uniformly distributed in respectively $[-2^{K-1}, -2^{K-1})$ and $[-2^{\widetilde{K}-1}, -2^{\widetilde{K}-1})$. Indeed, these inputs are base-2^K decompositions of LWE ciphertexts or fresh GSW ciphertexts, and any computational bias against the uniform distribution would form an attack on these schemes. In other words, if we randomize `bivRLWE` ciphertexts during normalization (e.g. we can always mask them with random ciphertexts of zero), with an overwhelming probability $1 - \varepsilon$ the arithmetic just needs to handle elements of size $B'_{\texttt{avg}} = C_\varepsilon \sqrt{2\ell N} 2^{K+K'-2}$ instead of the worst case $B'_{\texttt{worst}} = 2\ell N 2^{K+K'-2}$. We will use $C_\varepsilon = 17$ in this section, that corresponds to an error probability $\varepsilon < 2^{-40}$.

We present two equivalently good ways of handling such arithmetic: floating point or fixed-point approximations of the continuous FFT on backends whose mantissa can store B', or NTT over a friendly modulus larger than B'. The underlying arithmetic must be sufficiently atomic to be considered as native operations, therefore we will limit ourselves to 64-bit or 128-bit B'.

In a nutshell, the main result is that each choice of backend is bound to a precision B' and lead to a maximal limb size K: for all practical HE dimensions, the `float64` backend corresponds to $K \leq 19$, the `float128` backend (or fixed-point backends via 104-bit arithmetic in `AVX_IFMA`) would correspond to $K \leq 49$, doing NTT over a 60-bits modulus corresponds to $K \leq 22$, and $K \leq 52$ for a 120-bit modulus. And the key takeaway is that the most important success factor for a backend to be FHE friendly is to support the largest machine-word arithmetic; it is much less important if that arithmetic is modulo a power of two (fixed-point FFT), modulo a user-friendly prime number (NTT), or floating point (FFT).

6.1 Floating point backends

The first choice when it comes to FFT on bounded numbers is floating point backends. This algorithm is well studied and has been successfully used in the `TFHE` library since its origins. A naïve application of the formula $B'_{\texttt{avg}} = C_\varepsilon \sqrt{2\ell N} 2^{K+\widetilde{K}-2}$ for $K = \widetilde{K}$ and provided that we can use the 52 bits of mantissa without loss would bound K around 19. However, a lot of intermediate floating point operations occur between the start of the FFT, the products, the inverse FFT and we have to guarantee that the final error remains bounded by 1/2 to be recoverable by rounding. We decided to treat the problem experimentally, by sampling uniformly random polynomials $A(X), B(X) \in \mathcal{R}$ with coefficients $\in [-2^{K-1}, 2^{K-1})$, multiplying these two polynomials via $C = iFFT(FFT(A) * FFT(B))$ using 64-bit double-precision floats, and measuring the amplitude and the standard deviation of the error $C - AB$, as a function of N and K. From these measurements, we estimate the probability that a sum of such terms satisfy $\left\| \sum_{i=1}^{2\ell} C_i - A_i B_i \right\|_\infty < 1/2$, which is sufficient to recover the actual result by rounding.

Lemma 6.1. *Let $\ell, N \in \mathbb{N}$ and $\varepsilon > 0$. The maximal value of σ such that with probability $\geq 1 - \varepsilon$, the sum $v = \sum_{i=1}^{2\ell} v_i$ of 2ℓ independent real vectors $v_i \in \mathbb{R}^N$ with independent Gaussian coordinates of mean 0 and stdev σ is bounded by $\|v_\infty\| < 1/2$ is $\sigma \leq 1/\left(\sqrt{16\ell}\,\mathtt{erfinv}\left((1-\varepsilon)^{1/N}\right)\right)$.*

For a target probability error $\varepsilon = 2^{-40}$, $N = 65536$ and $\ell = 100$, the maximum σ is 0.00414. Experimentally we have observed that for those values the largest limb size we can choose is $K = 19$ for 64-bit words. We obtain the same result with $N = 32768$. For completeness, we have run the same experiment with 128-bit words (`float128`) that have a mantissa of 112 bits. In this case we see that we can choose $K = 48$, however the obtained σ is very close and we could even choose $K = 49$ given that ℓ is going to be much smaller than 100 due to the constraints outlined in Table 1.

Table 3 shows the results of these experiments for different values of K and N for 64-bit (`double`) and 128-bit (`float128`) words. The objective is to find the largest value of K for a given error boundary.

Unfortunately, to this date, `float128` is not a primitive type on x86 architectures and it's not available in all targets either. GCC supports `float128` operations via the quad-precision math library `libquadmath`, which is shipped along since version 4.6 of GCC, however, `float128` operations provide a poor performance compared to AVX accelerated doubles. Our experiment shows that due to the increased limb size K, large precision floats have potential, but without any dedicated hardware support, quad floats are not performant enough for our homomorphic backends. As an opening, we could however investigate the newer `AVX-IFMA` extensions set, whose instructions are already available on most recent commodity CPUs. These instructions allow to easily emulate 104-bit fixed-point arithmetic, and seems to be a perfect hardware-accelerated candidate for FFT computations, on a large limb sizes.

Table 3. Experimental standard deviation of floating point errors after a sum of FFT products for given N and varying K with 64 and 128-bit floats.

N	64-bit representation					128-bit representation					
64K	17	18	**19**	20	21	47	48	**49**	50	51	K
	0.0002	0.0008	**0.0031**	0.0124	0.0498	0.0003	0.0011	**0.00416**	0.0169	0.067	σ
32K	17	18	**19**	20	21	47	48	**49**	50	51	K
	0.0001	0.0005	**0.0021**	0.0085	0.034	0.0002	0.0007	**0.0028**	0.0112	0.0448	σ

6.2 NTT backends over a fixed modulus

As an alternative to FFT over the complex numbers, it is well known that cyclotomic multiplications can also be carried out by NTT, which has been the default choice of in the Full-RNS representation and later in [26]. All we need is a choice of one modulus or a product of moduli larger than B' that are NTT-friendly (unlike the frontend-ones in the CRT-representation) and replace all DFTs by NTT. NTT over moduli that are products of 30-bit or 60-bit primes can be accelerated on processors supporting `AVX2` extensions of `x86` architectures. Note that due to the overhead necessary for modular reductions as well as the missing support for native 64-bit SIMD integer multiplication on `AVX2`[5], one NTT on a 30-bit modulus requires the same number of clock cycles as one FFT on 64-bit floating points. Similarly, all NVIDIA GPUs provide native support for only 32-bit SIMT integer multiplication, thus, requiring emulation for larger (64-bit or more) SIMT integer multiplication.

Therefore, the best trade-off we have under the above constraints is to target 120-bit modulus NTT since the larger supported limb size $K = 52$ reduces the number of elementary operations in an external product: 120-bit NTT with $K = 52$ that has approximately the same running time as the 64-bit floating point FFT counterpart with $K = 19$ (the evidence for the latter can be deduced by combining the data from Table 1 with the micro-benchmarks from Table 4 below).

Comparative micro-benchmarks between floating point and NTT elementary operations are provided in Table 4: one DFT/iDFT operation is either an FFT or an NTT on a consecutive array of N elements. The `float64` and `float128` FFT operations are self-explanatory, whereas in 60-bit or 120-bit NTT, an element x is represented by two or four (lazily-reduced) 64-bit integers equal to x modulo (q_1, q_2) or (q_1, q_2, q_3, q_4), respectively. One SIMD `addmul`/twiddle consists of either one operation $r = r + ab$ over vectors in $\mathbb{C}^{N/2}$ or $(\mathbb{Z}/Q\mathbb{Z})^N$ or one twiddle-factor $(a, b) \to (a+\omega\cdot b, a-\omega\cdot b)$ where ω is a fixed (general) root of unity, whichever is slower. For $N = 32\text{K}$ or 64K, the twiddle factor is in general 10% faster than the `addmul`, despite the fact that the it contains one more subtraction, which indicates that these operations are memory-bound. `float64`, and the two NTTs use `AVX2` instructions, whereas `float128` is powered by `libquadmath` and does not benefit from any particular acceleration except for automorphisms that use only copy and sign-bit flipping.

We also provide benchmarks of homomorphic elementary operations in Table 5. We used the same parameter sets: $(N = 65536, L = 1729)$ and $(N = 32768, L = 865)$ as in [26]. The fast CKKS-RNS benchmarks have been run from the source code provided in [26] on the same machine as our own benchmarks. We evaluated the performance on the same machines as in Section 4:

[5] Originally, `AVX2` was supporting mainly floating point operations with 32-bit floats and 64-bit doubles - it was much later that integer operations were introduced. Today, not all 64-bit integer operations are supported - e.g., it was only recently that SIMD multiplications of vectors of 64-bit integers were introduced in `AVX512` and recently, Intel has disabled `AVX512` in AlderLake processors.

one n2-standard GCP instance with 64GB of RAM, and one 12-th Gen i7-1260p laptop with 64GB of RAM (Top and Down parts of Table 5). We expect that the performance numbers in these tables will be in constant evolution, as new hardware tend to support larger precision arithmetic. However, unlike what was commonly believed so far, any chip or device that can: either efficiently approximate the complex FFT with reasonable precision, or execute NTT on even just one single NTT-friendly modulus of reasonable size, is suitable for fast homomorphic computations at any depth. The efficiency of such device is directly related to the number of bits of mantissa or modulus it natively supports.

Table 4. DFT and SIMD arithmetic operations on a Xeon 2.8GHz n2-standard with 64GB of RAM instance. All benchmarks are single core.

backend	`float64` FFT		`float128` FFT		60-bit NTT		120-bit NTT	
K	19		49		22		52	
N	64K	32K	64K	32K	64K	32K	$64k$	$32k$
DFT/iDFT	125μs	57μs	60.3ms	28.4ms	534μs	243μs	1342μs	541μs
SIMD addmul/twiddle	27μs	9μs	2.17ms	1.07ms	49μs	28μs	93μs	48μs
automorphism	58μs	29μs	68μs	33μs	68μs	33μs	92μs	45μs

Table 5. Total running time per homomorphic operation over RLWE ciphertexts: CRT representations for full-RNS and [26], bivariate representations in our case. We recall that we use $L = 1729$ and $L = 865$ as in the implementation provided by the authors of [26].

Operation	Keyswitch		Automorphism	CKKS product
Size	$N = 64k$ L=1729	$N = 32k$ L=865	$N = 64k$ L=1729	$N = 64k$ L=1729
Hardware	n2-standard VM Xeon(R) CPU @ 2.8GHz, 64GB RAM			
- Full-RNS (best r)	3.111s	0.359s	3.279s	3.311s
- [26] (best r)	0.965s	0.161s	1.134s	1.155s
- ours: biv + fft-f64 ($K = 19$)	0.589s	0.086s	0.602s	0.862s
- ours: biv + ntt120 ($K = 52$)	0.541s	0.073s	0.547s	0.777s
Hardware	Laptop with Intel Core i7-1260P @ 4.7GHz, 64GB RAM			
- Full-RNS (best r)	1.598s	0.192s	1.796s	1.759s
- [26] (best r)	0.521s	0.085s	0.578s	0.598s
- ours: biv + fft-f64 ($K = 19$)	0.228s	0.027s	0.233s	0.335s
- ours: biv + ntt120 ($K = 52$)	0.218s	0.029s	0.221s	0.314s

Conclusion

In this paper we extend the key decomposition techniques from [26] by using a simpler and more natural base-2^K representation. It allows a better understanding of the parametrization of `TFHE`, `CKKS` and `BFV` schemes, which in turn speeds-up not only the main operations in `CKKS` and `BFV` schemes, but also low-depth computations in `TFHE`. In the Appendix D we also discuss how this speedup would impact the `CKKS` bootstrapping.

References

1. DARPA:data protection in virtual environments (DPRIVE).
2. TFHE: Fast Fully Homomorphic Encryption over the Torus. `https://tfhe.github.io/tfhe/`.
3. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
4. J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In R. Avanzi and H. Heys, editors, *Selected Areas in Cryptography – SAC 2016*, pages 423–442, Cham, 2017. Springer International Publishing.
5. J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. Cryptology ePrint Archive, Paper 2020/1203, 2020. `https://eprint.iacr.org/2020/1203`.
6. C. Boura, N. Gama, M. Georgieva, and D. Jetchev. CHIMERA: combining ring-lwe-based fully homomorphic encryption schemes. *J. Math. Cryptol.*, 14(1):316–338, 2020.
7. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In S. Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.
8. R. Cammarota. Intel HERACLES: homomorphic encryption revolutionary accelerator with correctness for learning-oriented end-to-end solutions. In F. Regazzoni and M. van Dijk, editors, *Proceedings of the 2022 on Cloud Computing Security Workshop, CCSW 2022, Los Angeles, CA, USA, 7 November 2022*, page 3. ACM, 2022.
9. H. Chen, I. Chillotti, and Y. Song. Improved bootstrapping for approximate homomorphic encryption. 11477:34–54, 2019.
10. J. H. Cheon, K. Han, and M. Hhan. Faster homomorphic discrete fourier transforms and improved fhe bootstrapping. Cryptology ePrint Archive, Paper 2018/1073, 2018. `https://eprint.iacr.org/2018/1073`.
11. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018*, volume 10820 of *Lecture Notes in Computer Science*, pages 360–384. Springer, 2018.
12. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. A full rns variant of approximate homomorphic encryption. *Selected areas in cryptography : ... annual international workshop, SAC ... proceedings. SAC*, 11349:347–368, 2018.
13. J. H. Cheon, A. Kim, M. Kim, and Y. S. Song. Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017.
14. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT 2016, Proceedings, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, 2016.
15. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1):34–91, 2020.

16. J. Cooley and J. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.
17. M. Creeger. The rise of fully homomorphic encryption: Often called the holy grail of cryptography, commercial FHE is near. *ACM Queue*, 20(4):39–60, 2022.
18. K. Derya, A. C. Mert, E. Öztürk, and E. Savas. Coha-ntt: A configurable hardware accelerator for ntt-based polynomial multiplication. *Microprocess. Microsystems*, 89:104451, 2022.
19. L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015, Proceedings, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, 2015.
20. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.
21. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.
22. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. *IACR Cryptol. ePrint Arch.*, page 99, 2012.
23. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013.
24. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO 2013, Proceedings, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, 2013.
25. S. Halevi, Y. Polyakov, and V. Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. *IACR Cryptol. ePrint Arch.*, page 117, 2018.
26. M. Kim, D. Lee, J. Seo, and Y. Song. Accelerating HE operations from key decomposition technique. *preprint available at* `https://eprint.iacr.org/2023/413.pdf`, 2023.
27. S. Kim, M. Park, J. Kim, T. Kim, and C. Min. Evalround algorithm in ckks bootstrapping. Cryptology ePrint Archive, Paper 2022/1256, 2022. `https://eprint.iacr.org/2022/1256`.
28. I. Kundu, E. Cottle, F. Michel, J. Wilson, and N. New. The dawn of energy efficient computing: Optically accelerating the fast fourier transform core. In *Photonics in Switching and Computing 2021*. Optica Publishing Group, 2021.
29. Y. Lee, S. Heo, S. Cheon, S. Jeong, C. Kim, E. Kim, D. Lee, and H. Kim. Hecate: Performance-aware scale optimization for homomorphic encryption compiler. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 193–204, 2022.
30. A. C. Mert, E. Öztürk, and E. Savas. FPGA implementation of a run-time configurable ntt-based polynomial multiplication hardware. *Microprocess. Microsystems*, 78:103219, 2020.
31. D. T. Nguyen and K. Gaj. Fast falcon signature generation and verification using armv8 neon instructions. In N. El Mrabet, L. De Feo, and S. Duquesne, editors, *Progress in Cryptology - AFRICACRYPT 2023*, pages 417–441, Cham, 2023. Springer Nature Switzerland.

32. Ö. Özerk, C. Elgezen, A. C. Mert, E. Öztürk, and E. Savas. Efficient number theoretic transform implementation on GPU for homomorphic encryption. *J. Supercomput.*, 78(2):2840–2872, 2022.
33. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.
34. S. S. Roy, F. Turan, K. Järvinen, F. Vercauteren, and I. Verbauwhede. Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. In *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*, pages 387–398. IEEE, 2019.
35. A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292, 1971.
36. F. Turan, S. S. Roy, and I. Verbauwhede. HEAWS: an accelerator for homomorphic encryption on the amazon AWS FPGA. *IEEE Trans. Computers*, 69(8):1185–1196, 2020.
37. E. R. Türkoglu, A. S. Özcan, C. Ayduman, A. C. Mert, E. Öztürk, and E. Savas. An accelerated GPU library for homomorphic encryption operations of BFV scheme. In *IEEE International Symposium on Circuits and Systems, ISCAS 2022, Austin, TX, USA, May 27 - June 1, 2022*, pages 1155–1159. IEEE, 2022.
38. Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. `https://github.com/zama-ai/tfhe-rs`.

A Normalization and reduction lemma proof

Proof of Lemma 3.3. Let $n = \lceil (L+M)/K \rceil$ be the number of algorithm iterations. Let $\texttt{acc}^{(i)}$ be accumulator value after step 3 of iteration i and let $\texttt{acc}^{(n+1)} = 0$. At iteration i, $n \geq i \geq 1$, we have:

$$\texttt{acc}^{(i)} = A_i + \epsilon_{i+1} \text{ and } R_i = \texttt{acc}^{(i)} - \epsilon_i 2^K, \tag{9}$$

here $\epsilon_i = \left\lfloor \texttt{acc}^{(i)} 2^{-K} \right\rfloor$ and has integer values.

The evaluation of result polynomial at 2^{-K} gives:

$$\phi_K(R) = \sum_{i=1}^{\ell} R_i 2^{-iK} = \sum_{i=1}^{\ell} \left(A_i + \epsilon_{i+1} - \epsilon_i 2^K\right) 2^{-iK}$$

Expanding the sum and simplifying common expressions we obtain:

$$\phi_K(R) = \sum_{i=1}^{\ell} A_i 2^{-iK} + \epsilon_{\ell+1} 2^{-\ell K} - \epsilon_1$$

Now, we will prove that $\|(\phi_K(A) - \phi_K(R)) \bmod \mathcal{R}\|_\infty$ is smaller than 2^{-L}. We have:

$$\phi_K(A) - \phi_K(R) = A_0 + \sum_{\ell < i} A_i 2^{-iK} - \epsilon_{(\ell+1)} 2^{-\ell K} + \epsilon_1.$$

Observe that terms A_0 and ϵ_1 are integers and are reduced by $\bmod \mathcal{R}$ operation, we obtain:

$$(\phi_K(A) - \phi_K(R)) \bmod \mathcal{R} = \sum_{\ell < i} A_i 2^{-iK} - \epsilon_{\ell+1} 2^{-\ell K}.$$

From (9) it is easy to see that $A_i - \epsilon_i 2^K \equiv R_i - \epsilon_{i+1}$. Using previous relations we have:

$$\begin{aligned}
(\phi_K(A) - \phi_K(R)) \bmod \mathcal{R} = & \\
&= 2^{-(\ell+1)K}(A_{\ell+1} - \epsilon_{\ell+1} 2^K) + \sum_{\ell+1 < i} A_i 2^{-iK} \\
&= 2^{-(\ell+1)K} R_{\ell+1} + \sum_{\ell+1 < i} A_i 2^{-iK} - \epsilon_{\ell+2} 2^{-(\ell+1)K} \\
&= \dots \\
&= \sum_{i=\ell+1}^{n} R_i 2^{-iK} + \sum_{n < i} A_i 2^{-iK}
\end{aligned}$$

Looking at the infinity norm we have:

$$\begin{aligned}\left\|(\phi_K(A) - \phi_K(R)) \bmod \mathcal{R}\right\|_\infty &= \left\| \sum_{i=\ell+1}^{n} R_i 2^{-iK} + \sum_{n<i} A_i 2^{-iK} \right\|_\infty \\ &< \left\| \sum_{i=\ell+1}^{n} 2^{-iK+K} + \sum_{n<i} 2^{-iK+M} \right\|_\infty \\ &\leq 2^{-\ell K+1} \leq 2^{-L}\end{aligned}$$

Element-wise operations are on integers in interval $\left(-2^{M+1}, 2^{M+1}\right)$

The accumulator variable `acc` (at step 3) has the largest values during algorithm execution. We will prove that its magnitude (i.e. infinity norm) never exceeds 2^{M+1}.

Algorithm step 3 increases accumulator value by at most 2^M and step 5 divides the new value by 2^K. After the first iteration, we have $\left\|\mathtt{acc}^{(n)}\right\|_\infty < 2^M$, after second iteration $\left\|\mathtt{acc}^{(n-1)}\right\|_\infty < 2^{M-K} + 2^M$ and so on until the last iteration where we have $\left\|\mathtt{acc}^{(1)}\right\|_\infty < \sum_{0\leq i<n} 2^{M-iK}$, which is the maximum accumulator magnitude attained during algorithm execution. We can rewrite the last expression as:

$$\left\|\mathtt{acc}^{(1)}\right\|_\infty < \sum_{0\leq i<n} 2^{M-iK} = 2^M \cdot \sum_{0\leq i<n} 2^{-iK} < 2^M \cdot 2 = 2^{M+1},$$

which proves the accumulator bound.

Complexity Algorithm steps 3-5 are executed $\lceil (L+M)/K \rceil$ times. In each iteration 5 operations are performed: an addition (step 3), 3 shifts (2 in step 4 and 1 in step 5) and a subtraction (step 4). The overall complexity of the algorithm is $5 \cdot \lceil (L+M)/K \rceil$ element-wise operations.

□

B Normalization and Conversion of Limb Sizes

In this section we introduce Algorithm 2, a general conversion and normalization with two limb size K and $\widetilde{K}$. This algorithm is a slightly more complex than Algorithm 1 with $K = \tilde{K}$, since it must handle additional binary shifts to synchronize the limb sizes, and thus, the for loop on a single index k is replaced by two while loops and two indexes $k, \tilde{k}$ that decrease at their respective speed. Besides that, it follows the same principle as the single-limb normalization.

C Optimized internal products in `BFV` and `CKKS`

Here, we provide some of the details on the optimizations as well as the complexity analysis (Theorem 3.2) of the efficient computation of the internal product from Section 3.5.

Algorithm 2 Conversion, Normalization and Reduction (from K to $\widetilde{K}$).

```
Input: A target precision of $L$ bits, an input limb size $K$ and an output limb size $\widetilde{K}$
Input: An input polynomial $A(X,Y) = \sum_{k\in\mathbb{Z}_{\geq 0}} A_k(X)Y^k$ satisfying $\|A\|_\infty \leq 2^B$
Output: A $\widetilde{K}$-normalized and reduced polynomial $R(X,Y)$ of degree $\leq \lceil L/\widetilde{K}\rceil$ in $Y$
    such that $\|\phi_{\widetilde{K}}(R) - \phi_K(A)\|_\infty \leq 2^{-L}$.
 1: $k = \lceil (L+B)/K\rceil$, $\widetilde{k} = \lceil (L+B)/\widetilde{K}\rceil$
 2: $\texttt{acc}(X) = \lfloor A_k(X)\cdot 2^{\widetilde{k}\widetilde{K}-kK}\rceil$
 3: while $\widetilde{k} \geq 1$ do
 4:     while $(\widetilde{k}-1)\widetilde{K} < (k-1)K$ do
 5:         $\texttt{acc}(X) \leftarrow \texttt{acc}(X) + A_{k-1}(X)\cdot 2^{(k-1)K-(\widetilde{k}-1)\widetilde{K}}$
 6:         $k \leftarrow k-1$
 7:     end while
 8:     $R_{\widetilde{k}}(X) \leftarrow \texttt{centermod}_{2^{\widetilde{K}}}(\texttt{acc}(X))$
 9:     $\texttt{acc}(X) \leftarrow (\texttt{acc}(X) - R_{\widetilde{k}}(X))/2^{\widetilde{K}}$
10:     $\widetilde{k} \leftarrow \widetilde{k}-1$
11: end while
12: Return $R(X,Y) = \sum_{k=1}^{\lceil L/K\rceil} R_k(X)Y^k$
```

Recall that the $*$ operator is the multiplication over $\mathbb{R}_N[X]$ and that the coefficients of normalized and reduced bivariate representations of (u,v) are automatically lifted to the real interval $[-1/2,1/2)$ as $(\tilde{u},\tilde{v})$. Unlike for CRT representations, the rounding to the nearest multiple of $2^{-L_{in}}$ at the start of the CKKS product (3) is easy to achieve in the bivariate base-2^K representation of the coefficients by simply truncating the representation to degree $\ell = \lceil L_{\text{in}}/K\rceil$ and rounding the last limb to the nearest multiple of $2^{L_{in} \bmod K}$. At the end of the products $\otimes_\star$ for $\star = \{\texttt{BFV}, \texttt{CKKS}\}$, the multiplication by the small integers p (in the case of BFV) or the large power of two $2^{L_{in}}$ (for CKKS) are also straightforward to express in limbs. The major part of the computation remains the multiplication $*$ over $\mathbb{R}_N[X]$. Fortunately, as already mentioned in Section 3.5, the bivariate representation is a ring isomorphism between $\mathcal{R}[Y]$ and $\mathbb{R}_N[X]$, so the product of two bivariate representations corresponds to the product in $\mathbb{R}_N[X]$.

In the full-RNS that is based on CRT-representations, products are carry-less and have complexity $O(\ell N)$. Unfortunately, our bivariate representation do not have such an analogue, so we have to multiply two representations over $\mathbb{Z}[X,Y]/\langle X^N+1\rangle$ of degree $\leq \ell-1$ and obtain a result of degree $\leq 2\ell-2$. The fastest multiplication algorithms for such polynomials involve a double DFT on the variables X and Y and have asymptotic run-time $O(\ell N\log_2(\ell N)) = O(\ell N(\log_2\ell + \log_2 N))$, that is, a logarithmic factor above the carry-less multiplication. Note, however, that this operation is not the bottleneck: an internal product over ciphertexts require at least one external product in $O(\ell^2 N)$ for the relinearization, so asymptotically, an internal product over bivRLWE ciphertexts has the same cost as its underlying external relinearization product.

To multiply two K-normalized and reduced bivariate polynomials a and b of degree ℓ we first perform the DFTs on the X variable to obtain $N/2$ univariate polynomials $(A_1(Y), \ldots, A_{N/2}(Y))$ and $(B_1(Y), \ldots, B_{N/2}(Y))$, each in $\mathbb{C}_N[X]$ of degree $\leq \ell - 1$. We then simultaneously multiply each $A_i(Y) \times B_i(Y)$ to form $N/2$ univariate polynomials $(C_1(Y), \ldots, C_{N/2}(Y))$ of degree at most $2\ell - 2$ using the fastest algorithm in practice between the naïve multiplication, Karatsuba multiplication, or DFT over Y. This takes the form of a SIMD computation, where all the elementary steps necessary to compute the first product $C_1 = A_1 \times B_1$ are replicated coordinate-wise on a vector of $N/2$ complex coordinates for the other products. Finally, we apply the inverse DFT on X to recover the result.

The SIMD part of the computation can be sped-up by the observation that the `BFV` product requires computing only the ℓ terms of lower degree since higher degree terms in Y represent a negligible quantity. Similarly, in `CKKS`, the final multiplication by $2^{L_{in}}$ eliminates all the low-degree terms after reduction modulo $\mathbb{Z}$, so we only need the ℓ terms of higher degree. The initial DFTs only need to be computed once per input, and two of the final inverse DFTs and normalizations can be postponed until after the external product.

With these optimizations in mind, we are ready to prove Theorem 3.2 and thus, obtain the final complexities:

Proof of Theorem 3.2. All our inputs a_1, b_1, a_2, b_2 are given in coefficient space, so 4ℓ DFT's on the variable X are consumed to bring them in DFT-space. To compute P_0, P_1 and P_2, we apply three strategies:

Naive product: for one plaintext internal product we have exactly $\ell(\ell-1)/2$ `addmul` operations to obtain either the the ℓ coefficients of lowest degree (`BFV`) or the ℓ coefficients of highest degree (`CKKS`). In the internal homomorphic encryption we have 4 products (to compute P_0, P_1 and P_2). This yields a baseline of $4\ell(\ell-1)/2$ products.

Karatsuba product: The first iteration of Karatsuba is quite interesting, as it uses the fact that P_2 is in fact (close to) $(a_1+a_2) \otimes_\star (b_1+b_2) - P_1 - P_2$ so three products plus four additions/subtractions are needed instead of four products in the naive algorithm, thus, $3\ell(\ell-1)/2$ products and 4ℓ additions. Pursuing the recursion would end-up executing less multiplications than the naïve approach (the number of multiplications would be in $O(\ell^{\log_2 3})$), however, one machine word addition often has the same throughput as one multiplication (which is the case of floating point and int64 operations on a CPU), and if we take additions into account, the number of binary operations follows the recursion $U(\ell) = 1$ if $\ell = 1$, otherwise $3U(\ell/2) + 6\ell$, which is super-quadratic. Moreover, the subsequent recursive calls to Karatsuba algorithm cannot be performed in-place anymore which requires expensive memory allocation. At the end, Karatsuba has to be limited to a single iteration to achieve $3\ell(\ell-1)/2 + 4\ell$ operations.

FFT product: we evaluate each polynomial on at least $2\ell - 1$ points, which is a DFT in Y, multiply those evaluations and compute the inverse DFT. The most efficient choice is to take the $2\ell'$-th roots of unity, where ℓ' is the smallest power of two larger or equal to ℓ. Because A_i have degrees lower or equal to ℓ', each of the four direct DFT has $\ell' \log_2 \ell'$ twiddle factors, and each of the three output DFT that have degree 2ℓ have $\ell'(\log_2 \ell' + 1)$ twiddle factors. That is a total of $7\ell' \log_2 \ell' + 9\ell'$ operations. This count is exact when $\ell = \ell'$ is a power of two, however, in the other cases, we would in theory have to consider that ℓ' can grow up to 2ℓ. In practice, since both the input and output polynomials are padded with zero coefficients, we can smooth the resulting complexity by propagating these known zero positions across the twiddle factors and eliminate those that are trivial.

Once the products P_0, P_1, P_2 are computed, we need to inverse the DFT in X and normalize the result: for P_2, this normalization needs to happen before the external product, however for P_0 and P_1, this iDFT and normalization can be delayed and merged with the ones at the output of the external product, on the very final result. Therefore we count only 4ℓ additional DFT's. $\square$

Remark 1. We can decrease the number of DFTs in X from 8 to 6 for the internal product and from 3 to 2 for the external product under the assumption that the default representation for a `bivRLWE` ciphertext is in DFT space ($\texttt{bivRLWE}^{\texttt{DFT}}$) rather than in coefficient space. In the present case, since DFTs are not the limiting factor, it only has a small effect on the overall complexity and becomes less natural if the external product uses multiple limb sizes $K, \widetilde{K}$. Also, unlike the Full-RNS case where reduction mod Q can be done in NTT space as well, all the normalizations in this work and also in [26] require a round-trip to coefficient space.

D CKKS bootstrapping in the bivariate representation

In this section, we verify that the speed-ups obtained on external products, internal products and linear combinations presented on the bivariate representation are very likely to extend to more complex constructions, like the CKKS and BFV bootstrappings, without changing those algorithms.

CKKS bootstrapping [11] and improved variants [10,9,5,27] rely on the following steps, to bootstrap a high noise ciphertext (a, b), that encodes μ:

1. Start from the ciphertext $(\text{centerlift}(a)/2^L, \text{centerlift}(b)/2^L)$. By definition, it encodes $P(X) = \mu/2^L + I(X)/2^L$ where I is an integer polynomial of norm $\|I\|_\infty = O(\sqrt{N})$.
2. Apply the coeffsToSlots operation, which consists in multiplying the slots by the complex inverse FFT matrix modulo $X^{N/2} - i$. The slots now contain the coefficients of P.
3. Evaluate (an approximation of) $x \to 2^{-L}.\mathit{centermod}(2^L.x)$, to eliminate I, so that the slots contain the coefficients of $\mu.2^{-L}$

	$N = 65536,\ L = 1729$					$N = 32768,\ L = 865$				
K	ℓ	#DFTs	#SIMD prods	#IP DFTs	#IP SIMD prods	ℓ	#DFTs	#SIMD prods	#IP DFTs	#IP SIMD prods
13	133	399	35378	931	49058	67	201	8978	469	15053
16	109	327	23762	763	30278	55	165	6050	385	8927
19	**91**	**273**	**16562**	**637**	**22889**	**46**	**138**	**4232**	**322**	**7016**
22	79	237	12482	553	18683	40	120	3200	280	5600
25	70	210	9800	490	15908	35	105	2450	245	4287
28	62	186	7688	434	10640	31	93	1922	217	3203
31	56	168	6272	392	9161	28	84	1568	196	2744
34	51	153	5202	357	8037	26	78	1352	182	2366
37	47	141	4418	329	7211	24	72	1152	168	2016
40	44	132	3872	308	6635	22	66	968	154	1694
43	41	123	3362	287	5883	21	63	882	147	1543
46	38	114	2888	266	5054	19	57	722	133	1263
49	36	108	2592	252	4536	18	54	648	126	1134
52	**34**	**102**	**2312**	**238**	**4046**	**17**	**51**	**578**	**119**	**1011**

Table 6. Number of operations in one half-external product (i.e. one relinearization, one keyswitch, or one automorphism) and in one BFV/CKKS internal product (IP). Complete version of Table 1.

4. Apply slots2coeffs, which consists in multiplying the slots by the FFT matrix modulo $X^{N/2} - i$. It yields a low-noise ciphertext of $\mu.2^{-L}$.

Under the bivariate representation, step 1 is entirely free! It suffices to consider that all the most significant limbs are equal to zero.

Steps 2 and 4 can use the usual radix-r fast-FFT tradeoff, that decomposes the $N/2 \times N/2$ FFT matrix into a product of $\log_r(N/2)$ sparse matrices with at most $2r-1$ non-zero diagonals of norm 1. Evaluating one of such matrix product boils down to a public linear combination with 30-60-bits coefficients, between $2\sqrt{r}$ automorphisms, and the speed-up covered in Section 3.4 and Table 5 apply.

Step 3 usually remarks that the function we evaluate over the slots only needs to coincide with the centermod function at a very close proximity of exact multiples of 2^{-L}. In practice, the function $x \to \sin(2\pi 2^L.x)$ has been traditionally mentioned as a good candidate. It has a fast convergent Taylor Series, which gives a moderate degree polynomial approximation. It can even be factored as $Im(exp(2i\pi x)^{2^L})$, which allows further running-time vs. depth by repeated squaring of lower degree approximations. In general, evaluating a degree d polynomial boils down to $O(d)$ internal CKKS products at depth $O(\log_2(d))$, followed by public linear combinations, which remain negligible compared to the squaring effort. The performance and speed-up of the bivariate representation on underlying CKKS internal product is covered in Section 3.5 and Table 5.

Overall, complex constructions like CKKS (and BFV) bootstrappings are solely expressed in terms of automorphisms and internal products, a smaller amount of public linear combinations, and nothing else. These algorithms are agnostic of the backend arithmetic, and they are therefore very likely to be directly impacted by the same performance gains as those depicted in our benchmarks section.

E Forward noise propagation theorems

All the theorems about the external products in the main Section have been provided in a backward propagation direction, which is the most useful for compilation purposes: we consider the full plaintext circuit and assign nodes and levels to each elementary operation one by one starting from the end. Each time, we start from the targeted output noise, and deduce the (larger) minimal required noise levels for the products. Whenever the level become too high (i.e. not efficient, or noise too small to be secure) , we can place a bootstrapping and continue backwards at a lower level.

In a more traditional interpreted language scenario, we already have existing ciphertexts and pre-existing relinearization keys, that we must just feed-in as input to a homomorphic product, at their current level of noise. In this case, we can use forward propagation formulae to deduce the output noise. In a real circuit, these forward-propagation formulae end-up forming a large system of constraints that is usually non-linear, and are quite hard to optimize by fhe compilers, compared to their backward-propagation counterpart. However, they are in general also easier to understand for a human being, so we provide them in this section.

Also, the main section theorems use the worst-case propagation, based on the infinity norm of the noise. These formulae are unconditionally true, but predict a noise growth that is in bits twice as fast as the reality. As explained in Section 4, the practical noise growth can be easily obtained if we use the additional assumption that all `bivRLWE` noises are samples from independent Gaussian distributions of same variance, and they remain independent. Such independence can be enforced randomizing ciphertexts between each external or internal product. In practice, a very mild randomization are in general sufficient to turn exceptionally large correlations in dot products into the average-case behaviour.

This is the analogue of Theorem 3.1 with forward average-case and worst-case noise propagation:

Theorem E.1 (half external product - forward propatation (secret$\times$public)). *Let $K, \widetilde{K}$ be limb sizes and let $s \in \mathcal{R}$ a small key. Let C_u be a* $\texttt{bivHalfRGSW}^{\texttt{DFT}}_{s,K,\widetilde{K}}$ *of an integer polynomial $u \in \mathcal{R}$, composed of $\tilde{\ell}$ ciphertexts of ℓ limbs, satisfying one of these two bounds:*

- *noise variance V_{GSW}, (average case)*
- *noise infinity norm ε_{GSW} (worst-case)*

and let $a = \sum (A_i) Y^i \in \mathcal{R}[Y]^2$ be $\widetilde{K}$-normalized.

$$C_f \boxdot a = \texttt{normalizeReduce}\left(\texttt{iDFT}\left(\sum_{i=1}^{\tilde{\ell}} \texttt{DFT}(A_i) \cdot c_i\right)\right)$$

is a $\texttt{bivRLWE}_{s,K}$ *of* $u \cdot \phi_K(a)$ *with, depending on which input noise condition above is satisfied:*

- *output noise variance* $V_{out} \leq \tilde{\ell} N 2^{2\widetilde{K}-2} V_{GSW} + \|u\|_2^2 \varepsilon_{Dec}^2$, *(average case)*
- *noise infinity norm* $\varepsilon_{out} \leq \tilde{\ell} N 2^{\widetilde{K}-1} \varepsilon_{GSW} + \|u\|_1 \varepsilon_{Dec}$ *(worst-case)*

where $\varepsilon_{Dec} = 2^{-\widetilde{K}\tilde{\ell}-1}$ *when the degree of* a *is* $\leq \tilde{\ell}$ *(approx decomposition), otherwise zero (exact decomposition).*

Such computation requires

- $\tilde{\ell}$ *bounded DFT's of the* A_i'*s of norm* $\leq 2^{\widetilde{K}-1}$,
- $2\ell\tilde{\ell}$ *element-wise* $\texttt{addmul}$*'s in DFT domain for polynomials of norm* $\leq N\tilde{\ell}2^{\widetilde{K}+K-2}$,
- 2ℓ *bounded DFT's for the results* $C_j(X)$ *with norm bound* $N\tilde{\ell}2^{\widetilde{K}+K-2}$,
- 2ℓ *element-wise additions/shifts/masks to normalize and truncate the result.*

And this is the analogue of Corollary 3.2, much closer to the traditional presentation in [14], again with worst-case and average-case forward-propagation of noise.

Corollary E.1 (full external product- forward propagation (secret$\times$secret)). *Let* $K, \widetilde{K}_A, \widetilde{K}_B$ *be limb sizes, let* $u \in \mathcal{R}$ *be a small polynomial, let* $v \in \mathbb{T}_N[X]$ *be a message, let* $s \in \mathcal{R}$ *a small key, and*

- *Let* C_u *be a* $\texttt{bivHalfRGSW}^{\texttt{DFT}}_{s,K,\widetilde{K}_B}$ *of* $u \in \mathcal{R}$, *composed of* $\tilde{\ell}_B$ *ciphertexts of* ℓ *limbs, and of noise variance (resp. amplitude) bounded by* V_A *(resp.* ε_A*)*
- *Let* C_{su} *be a* $\texttt{bivHalfRGSW}^{\texttt{DFT}}_{s,K,\widetilde{K}_A}$ *of* $su \in \mathcal{R}$, *composed of* $\tilde{\ell}_A$ *ciphertexts of* ℓ *limbs, and of noise variance (resp. amplitude) bounded by* V_A *(resp.* ε_A*)*
- $(A, B) \in \mathcal{R}[Y]^2$ *is a* $\texttt{bivRLWE}_{s,K}$ *ciphertext, of noise variance (resp. amplitude) bounded by* V_{in} *(resp.* ε_{in}*).*

Then

$$(C_u, C_{su}) \boxdot (a, b) = \texttt{normalizeRed}\left(\texttt{iDFT}\left(\sum_{i=1}^{\tilde{\ell}_B} \texttt{DFT}(B_i) \cdot c_i - \sum_{i=1}^{\tilde{\ell}_A} \texttt{DFT}(A_i) \cdot d_i\right)\right),$$

is a $\texttt{bivRLWE}_{S,K,2^{-L}}$ *encryption of* $u \cdot v$. *Here,* A *and* B *have been* $\widetilde{K}_A$ *and* $\widetilde{K}_B$*-normalized, and truncated to degree respectively* $\tilde{\ell}_A$ *and* $\tilde{\ell}_B$.

Let ε_{out} *and* V_{out} *denote the amplitude and variance of the output ciphertext, these bounds are satisfied:*

$$\varepsilon_{out} \leq \varepsilon_{in} + \tilde{\ell}_A N 2^{\widetilde{K}_A-1} \varepsilon_A + \|us\|_1 2^{-\tilde{\ell}_A\widetilde{K}_A-1} + \tilde{\ell}_B N 2^{\widetilde{K}_B-1} \varepsilon_B + \|u\|_1 2^{-\tilde{\ell}_B\widetilde{K}_B-1}$$
$$V_{out} \leq V_{in} + \tilde{\ell}_A N 4^{\widetilde{K}_A-1} V_A + \|us\|_2^2 4^{-\tilde{\ell}_A\widetilde{K}_A-1} + \tilde{\ell}_B N 4^{\widetilde{K}_B-1} V_B + \|u\|_2^2 4^{-\tilde{\ell}_B\widetilde{K}_B-1}$$

computing this encryption with $\ell = \lceil (L_2' + \log_2 N + 2)/K \rceil$ *requires at most*

- $\tilde{\ell}_A + \tilde{\ell}_B$ *bounded DFT's of the* A_i, B_i*'s of norm* $\leq 2^{\widetilde{K}-1}$,

- *$2\ell(\widetilde{\ell}_A + \widetilde{\ell}_B)$ element-wise* **addmul***'s in DFT domain for polynomials of norm $\leq 2N\widetilde{\ell}2^{\widetilde{K}+K-2}$,*
- *2ℓ bounded DFT's for the results $C_j(X)$ with norm bound $2N\widetilde{\ell}2^{\widetilde{K}+K-2}$,*
- *$\widetilde{\ell}_A+\widetilde{\ell}_B+2\ell$ element-wise additions/shifts or masks to normalize and truncate the input ciphertext and the final result.*

F Frontend and Large Number Arithmetic: Bivariate vs. CRT Representations

The main idea in both [26] and the present work is the decoupling of the cyclotomic arithmetic from the large number arithmetic in BFV, CKKS, TFHE and Chimera which, in turn, allows to fully benefit from the small polynomial coefficients produced by the gadget decomposition. In our work, the large number arithmetic corresponds to the bivariate representation we presented in Section 3, which, in essence, is a base-2^K decomposition into ℓ limbs with complexity $O(\ell \log_2 \ell)$. In [26], the large number arithmetic is performed via CRT representation of big numbers. The parametrization is a bit more complex to grasp since the various parameters $d, r, p_i, q_i, Q, \widetilde{Q}$ appear for historical reasons and for comparison to the former full-RNS algorithm. Yet, the parameters ℓ and B' have the same meaning as the ones in our work - large numbers of L bits are represented via their remainders modulo ℓ primes of size K bits each (which do not need to be NTT-friendly) and the arithmetic has complexity $O(\ell)$. Therefore, following the same steps as in Section 3, We could define the CRT representation of $\mathbb{T}_N[X]$ as:

- A sufficiently large family of K-bit prime numbers $(q_i)_{i\in I}$,
- The space of CRT-representations $(u_i)_i \in I \in \mathcal{R}^I$ (instead of $\mathcal{R}[Y]$), a representation of *degree* ℓ has support over $u_1, \ldots, u_\ell$ only, the rest is zero. A normalized representation satisfies $u_i \in [-(q_i-1)/2, (q_i-1)/2]$ (analogue to Lemma 3.1).
- An $\mathcal{R}$-module homomorphism $\phi^{\mathtt{crt}}\colon \mathcal{R}^I \to \mathbb{R}_N[X]$ (the evaluation homomorphism) given by $\phi^{\mathtt{crt}}((u_i)_{i\in I}) := \sum_{i\in I} u_i/q_i$. Any element of $\mathbb{T}_N[X]$ can be approximated at distance $\leq 2^{-K\ell}$ by a degree-ℓ representation (analogous to Lemma 3.3).
- crtRLWE of precision $2^{-K\ell}$ as RingLWE ciphertexts whose representation of coefficients have degree ℓ. (analogue to Definition 3.3)
- crtHalfRGSW of precision $2^{-K\ell}$ as a family of ℓ RingLWE ciphertexts that encrypt $1/q_i$ with precision $2^{-K\ell}$ (analogue to Definition 3.4).
- Half external products, full external products, automorphisms work the same on crtHalfRGSW and crtRLWE as in their bivariate counterparts and follow the same noise propagation formula as in Theorem 3.1, Corollary 3.2, Lemma 3.5, if we replace biv by crt. However, a few workarounds are needed to define the internal products because φ is not a ring morphism.

The four main differences between the bivariate representation and the CRT-representation are:

- The CRT-representation does not have the prefix property: a last digit removal procedure allows to recompute a weaker precision from a higher one, this computation is relatively cheap, but not free.
- The CRT representation only has proper subrings of precision 2^{-Ki} where i is integer: the CKKS product must round to an exact multiple of $1/\prod_{j=1}^{i} q_j$, whereas the bivariate representation can afford any rounding to any 2^{-L} bits of precision.
- There is one different product per degree for which ϕ^{crt} is a ring morphism: namely $x, y \to q_1, \dots, q_\ell.xy$ in degree ℓ. It happens to be the product we need for CKKS.
- If all half and full external products, keyswitches, relinearizations and automorphisms have exactly the same cost in both bivariate and CRT-cases at a given precision, internal CKKS or BFV products would be a little bit faster in the CRT representation, with a second order logarithmic difference in complexity: of $2\ell^2 N + 5\ell N + \circ(\ell N)$ in the CRT-case instead of $2\ell^2 N + 7\ell \log_2(\ell) N + \circ(\ell \log(\ell) N)$ in the bivariate case.

In both cases, bivariate and CRT, the cost of an homomorphic operation is dominated by the cost of a half external product in $O(\ell^2 N)$, that is included in keyswitches, automorphisms and in BFV/CKKS products. All the big-number arithmetic operations on K-bit limbs (whether these are base-2^K digits or residue modulo a K-bit prime) are carried out in parallel in DFT space on N-dimensions, which makes the two approaches equally parallelizable or vectorizable even if the big number arithmetic needs to propagate carries. Both approaches operate directly on normalized representations that come from a theoretical decomposition of $\mathbb{T}_N[X]$, and only require a linear number of DFTs per homomorphic operation (except the keygen that is done offline), which improve upon the quadratic number in Full-RNS. In the next section about the backend, we will see that for both the bivariate and the CRT approaches, these DFTs can be instantiated with similar performance, as either approximated FFT over the complex field, or NTT. And finally, both approaches require an additional normalization in coefficient space after every homomorphic operation to avoid unwanted wraparounds or overflows on B' bits.

We now point out some very important advantages of choosing the bivariate representation over the CRT representation: how the freedom of picking any input level L_{in} in the CKKS product allows to pack a larger number of homomorphic operations for a given level. We also show that the split of the full external product in two halves lead to an acceleration of TFHE bootstrapping, due to a better handling of the parameters.

F.1 Discrete vs. continuous levels in CKKS

The CKKS internal product formula (3) applies to all input bit precision parameters L_{in}: for any fixed plaintext mantissa precision ρ, feeding TRLWE ciphertexts with input noise rate $2^{-L_{\text{in}}}$ to (3) yields a fixed-point product of output noise rate $2^{-L_{\text{in}}+\rho+\log_2 N}$ that represents the ρ most-significant bits of the plaintext product. By varying L_{in}, we obtain any desired output precision L_{out}.

Since it is unrealistic to go back-and-forth between the original large integer coefficients and their full-RNS / CRT representations of `CKKS`, the number $2^{L_{\text{in}}}$ in (3) has to be replaced by one of the discrete products of $Q_i = q_1, \dots, q_i$ of the CRT basis that is close to $2^{L_{\text{in}}}$. In full-RNS, each q_i must be at least 18 bits to be NTT friendly for $N = 2^{16}$, and more generally, even though the CRT-representation in this section and in [26] raised the NTT-friendliness restriction, using a smaller bitsizes K make the scheme less efficient; in practice, it is usual to have K between 30-40 bits. If the noise of an input ciphertext is not close to an exact multiple of K-bits, (usually refered as half-multiplicative level), it has to be mod-rescaled to the next available one before a `CKKS` product can be attempted. E.g. after a cheap homomorphic operation like the sum of 1000 ciphertexts, or after computing a trace, or just after a small linear combination with coefficients between -30 and 30, the noise has only increased by 5 to 8 bits, and to be able to start the next CKKS product, we need to multiply/rescale the ciphertext by a factor 2^{K-8} to 2^{K-5} bits to come back to a (half-)level boundary. This represents a considerable loss of homomorphic budget.

Our bivariate approach does not have this limitation: any input noise level can be fed-in to Equation 3, or equivalently, any desired output noise level can be attained without any penalty on the noise overhead, that stays $\rho + \log_2(N)$ bits. A `bivRLWE` ciphertext at one given level of noise will be able to support more homomorphic operations before the noise is saturated. This statement is corroborated by the current progresses in FHE compilers: whereas the first CKKS compilers were operating on "full" 40-bit levels, the recent Hecate compiler by [29] was able to generate 27% more efficient CKKS circuits by considering "half-levels" of 20-bits. This line of work should continue to give more impressive results using the "continuous" single-bit levels and the simple noise propagation rule $L_{\text{out}} = L_{\text{in}} - \rho - \log_2 N$ during a product, and matches the `BFV` product.

While we are waiting for the next generation of CKKS compilers on continuous levels, our scheme is trivially backwards-compatible with CKKS compilers that support discrete full-levels or half-levels only: we just need to set $L_{\text{in}} = 40.\text{level}$, and as a bonus, the prefix-property of our representation implicitly relinearizes the input ciphertext for free!

F.2 Scheme switching: across schemes and representations

Chimera [6] introduced a few years ago a scheme switching concept that would allow to switch back and forth between `TFHE`, `BFV` and `CKKS` representations, however it did not provide a solution to efficiently deal with big numbers. At that time, the construction remained under the assumption that it would be efficient to do efficient multiprecision fixed-point FFT, and none of the experiments with GMP and MPFR provided fast enough results for levels > 2 compared to the full-RNS approach.

In this paper, the bivariate representation (which we see as a direct consequence of [26] second gadget decomposition), is in some sense the missing piece that allows to instantiate Chimera's scheme switching efficiently in practice.

In addition, the `HalfRGSW` product can efficiently convert not only between schemes, but also back and forth between the `crtRLWE` and `bivRLWE` ciphertexts, preserving the noise level and the plaintext: in the CRT to bivariate direction, all we need is a family of K-normalized representations of each $1/q_i$, and in the bivariate to CRT direction, a family of normalized CRT-representations of $1/2^{-Ki}$, both with precision $2^{-K\ell}$. In both cases, Theorem 3.1 fulfills such conversion in time $O(\ell^2 N)$.

This makes very efficient bridges between not only `TFHE`, `CKKS` and `BFV` ciphertexts, but for each scheme, between their CRT and bivariate representations.