Homomorphic Multiple Precision Multiplication for CKKS and Reduced Modulus Consumption

Jung Hee Cheon
CryptoLab Inc.
Seoul National University
Seoul, Republic of Korea
jhcheon@snu.ac.kr

Wonhee Cho
Seoul National University
Seoul, Republic of Korea
wony0404@snu.ac.kr

Jaehyung Kim
CryptoLab Inc.
Seoul, Republic of Korea
jaehyungkim@cryptolab.co.kr

Damien Stehlé
CryptoLab Inc.
Lyon, France
damien.stehle@cryptolab.co.kr

ABSTRACT

Homomorphic Encryption (HE) schemes such as BGV, BFV, and CKKS consume some ciphertext modulus for each multiplication. Bootstrapping (BTS) restores the modulus and allows homomorphic computation to continue, but it is time-consuming and requires a significant amount of modulus. For these reasons, decreasing modulus consumption is crucial topic for BGV, BFV and CKKS, on which numerous studies have been conducted.

We propose a novel method, called Mult^2, to perform ciphertext multiplication in the CKKS scheme with lower modulus consumption. Mult^2 relies an a new decomposition of a ciphertext into a pair of ciphertexts that homomorphically performs a weak form of Euclidean division. It multiplies two ciphertexts in decomposed formats with homomorphic double precision multiplication, and its result approximately decrypts to the same value as does the ordinary CKKS multiplication. Mult^2 can perform homomorphic multiplication by consuming almost half of the modulus.

We extend it to Mult^t for any $t \geq 2$, which relies on the decomposition of a ciphertext into t components. All other CKKS operations can be equally performed on pair/tuple formats, leading to the double-CKKS (resp. tuple-CKKS) scheme enabling homomorphic double (resp. multiple) precision arithmetic.

As a result, when the ciphertext modulus and dimension are fixed, the proposed algorithms enable the evaluation of deeper circuits without bootstrapping, or allow to reduce the number of bootstrappings required for the evaluation of the same circuits. Furthermore, they can be used to increase the precision without increasing the parameters. For example, Mult^2 enables 8 sequential multiplications with 100 bit scaling factor with a ciphertext modulus of only 680 bits, which is impossible with the ordinary CKKS multiplication algorithm.

CCS CONCEPTS

• **Security and privacy → Cryptography**.

KEYWORDS

Fully Homomorphic Encryption, CKKS scheme, Approximate multiplication, High precision, Small parameters

1 INTRODUCTION

Homomorphic Encryption (HE) provides a method to protect data even during its processing. Since Gentry first opened the world of Fully Homomorphic Encryption (FHE) in [17], its performance has been improved dramatically. Among several FHE schemes, the Cheon–Kim–Kim–Song (CKKS) scheme proposed in [12] supports efficient real and complex number computations, which is essential when managing real-world data including those arising in machine learning [22, 25–30, 37].

In the CKKS scheme, the ciphertext modulus decreases as multiplications are performed. When the ciphertext modulus becomes too small, multiplications cannot be performed anymore. A solution is to bootstrap the ciphertext to re-increase the modulus. The CKKS bootstrapping [10] consists of a homomorphic evaluation of a polynomial approximation to the modular reduction function, combined with discrete Fourier transformations. Since homomorphic polynomial evaluation and linear transformations are costly, bootstrapping itself takes time and consumes a significant amount of ciphertext modulus.

For this reason, one is often faced with a shortage in ciphertext modulus, which leads to increasing the ciphertext dimensions and performing costly bootstrappings. In this sense, the ciphertext modulus has been considered an invaluable resource, and many works have focused on reducing modulus consumption. This includes the introduction of gadget decomposition [3, 5, 6, 20, 21] for the key switching steps notably used for homomorphic multiplication and bootstrapping.

In this work, we introduce a novel technique to reduce modulus consumption, called Mult^2. It almost halves the modulus consumption while maintaining the asymptotic time complexity (and sometimes improving it). Mult^2 enables more multiplications for somewhat homomorphic encryption (SHE) and allows to perform more multiplications before bootstrapping for FHE. We further support the theoretical analysis of Mult^2 with experiments.

1.1 Homomorphic Euclidean division and ciphertext decomposition

To reduce the modulus consumption in homomorphic multiplication (while maintaining the precision), we aim at decomposing the multiplication of large bit-size multiplicands into several multiplications of smaller bit-size multiplicands. This is motivated by the observation that modulus consumption is driven by the size of the multiplicands. Consider the multiplication of two $2k$-bit integers m_1 and m_2. We decompose each of them into two k-bit pieces as $m_i = 2^k \cdot \hat{m}_i + \check{m}_i$ for $i \in \{1, 2\}$. Then we have:

$$m_1 m_2 = 2^{2k} \hat{m}_1 \hat{m}_2 + 2^k (\hat{m}_1 \check{m}_2 + \check{m}_1 \hat{m}_2) + \check{m}_1 \check{m}_2.$$

The $2k$-bit multiplication can be decomposed into four k-bit multiplications.

This approach immediately faces two difficulties. First, we need a way to homomorphically decompose a large bit-size plaintext into smaller pieces. Second, starting from pairs, one multiplication results in three pieces ($\hat{m}_1\hat{m}_2$, $\hat{m}_1\check{m}_2 + \check{m}_1\hat{m}_2$, and $\check{m}_1\check{m}_2$), and the number of pieces keeps increasing with more multiplications.

Let us focus on the increase of the number of pieces. In CKKS, the underlying arithmetic is not over integers but over fixed-point approximations to real (or complex) numbers. Starting with multiplicands with relative precision 2^{-2k}, we are interested in the product only with relative precision $\approx 2^{-2k}$. Put differently, only the first two pieces, $\hat{m}_1\hat{m}_2$ and $\hat{m}_1\check{m}_2 + \check{m}_1\hat{m}_2$, are of interest, and the third one, $\check{m}_1\check{m}_2$, can be dropped. Hence the number of pieces remains constant. Note that relying on smaller precision arithmetic for fixed-point or floating-point arithmetic is a standard technique (see, e.g., [16, 23]).

We now consider the task of homomorphically decomposing a large bit-size plaintext into smaller pieces.

Attempt 1: decompose then encrypt. As a first attempt, we may try to decompose a large bit-size plaintext into a pair of smaller plaintexts in the clear, and encrypt them into separate ciphertexts. Let N be a power of two and Q a positive integer. Consider a plaintext polynomial $m \in R_Q := \mathbb{Z}_Q[x]/(x^N + 1)$. Now, given a positive integer k, we decompose m into $m = 2^k \hat{m} + \check{m}$ with $\check{m} := (m \bmod 2^k)$ and encrypt each component independently:

$$m \longrightarrow (\hat{\mathsf{ct}}, \check{\mathsf{ct}}), \text{ where } \hat{\mathsf{ct}} = \mathsf{Enc}(\hat{m}) \text{ and } \check{\mathsf{ct}} = \mathsf{Enc}(\check{m}).$$

However, note that CKKS encryption adds noise to the plaintexts. We have:

$$2^k \mathsf{Dec}(\hat{\mathsf{ct}}) + \mathsf{Dec}(\check{\mathsf{ct}}) = 2^k (\hat{m} + \hat{e}) + (\check{m} + \check{e})$$

for some $\hat{e}$ and $\check{e}$ in $R := \mathbb{Z}[x]/(x^N + 1)$, with $\|\hat{e}\|_\infty, \|\hat{e}\|_\infty$ small. To fix the ideas, we can even consider them to be tiny compared to 2^k. Even though $\hat{e}$ and $\check{e}$ are small, as $\hat{e}$ is multiplied by 2^k, the error it induces leads to an overall relative precision of only $\approx k$ bits. In particular, the error $2^k \hat{e}$ is larger than $\check{m}$.

Attempt 2: encrypt then decompose. Instead, we propose to homomorphically decompose the plaintext underlying a ciphertext. Note that computing $\hat{m}$ and $\check{m}$ from m is exactly a Euclidean division. Divisions are notoriously difficult to implement homomorphically, but we show how to achieve a weak form of it when the divisor divides the ciphertext modulus. Consider a CKKS ciphertext $\mathsf{ct} = \mathsf{Enc}(m) \in R^2_{2^k Q}$, for modulus $2^k Q$ divisible by the divisor 2^k (the approach works for divisors q_{div} of arbitrary arithmetic shapes, but here we keep 2^k for consistency with the above discussions). We perform a (component-wise) Euclidean division by 2^k on ct, obtaining its quotient $\hat{\mathsf{ct}}$ and its remainder $\check{\mathsf{ct}}$:

$$\mathsf{DCP}_{2^k}(\mathsf{ct} = 2^k \cdot \hat{\mathsf{ct}} + \check{\mathsf{ct}}) \ := \ (\hat{\mathsf{ct}}, \check{\mathsf{ct}}) \in R^2_Q \times R^2.$$

Since the absolute value of each coefficient of $\check{\mathsf{ct}}$ is $\leq 2^k/2$, we can consider it as an element of R^2_Q when $Q > 2^k$.

If $m = 2^k \hat{m} + \check{m}$ as above, it is entirely possible that we do not have $\mathsf{Dec}(\hat{\mathsf{ct}}) = \hat{m}$ and $\mathsf{Dec}(\check{\mathsf{ct}}) = \check{m}$. However, since the decryption function is $\mathbb{Z}$-linear, we have $m = \mathsf{Dec}(ct) = 2^k \mathsf{Dec}(\hat{\mathsf{ct}}) + \mathsf{Dec}(\check{\mathsf{ct}})$. Hence if we write $\mathsf{Dec}(\hat{\mathsf{ct}}) = \hat{m} + \hat{e}$, then $\mathsf{Dec}(\check{\mathsf{ct}}) = \check{m} - \check{e}$ with $\check{e} = 2^k \hat{e}$. Now, as $\|\check{\mathsf{ct}}\|_\infty \leq 2^k/2$ and decryption is an inner product over R by a small-coefficient secret key vector, we have that $\mathsf{Dec}(\check{\mathsf{ct}}) = \check{m} - \check{e}$ is small. This implies that the error $\check{e}$ cannot be much larger than 2^k, and hence $\hat{e} = 2^{-k} \check{e}$ must be small. As the quotient error $\hat{e}$ is small and the remainder is consistent with the somewhat erroneous quotient, this gives a weak form of homomorphic Euclidean division, where the quotient may lead to a small remainder but maybe not the smallest remainder possible.

Now that the first two difficulties are handled, our goal is to define a homomorphic multiplication for decomposed ciphertexts.

1.2 Multiplication of decomposed ciphertexts

The CKKS homomorphic multiplication consists of three steps: tensoring (Tensor), relinearization (Relin), and rescaling (RS), as recalled in Section 2.2. We adapt each of them to decomposed ciphertexts to obtain a Mult^2 homomorphic multiplication algorithm on ciphertexts given in decomposed forms.

Tensoring. We define a Tensor^2 operation, also denoted by $\otimes^2$, on decomposed ciphertexts $\mathsf{ct}_1 = (\hat{\mathsf{ct}}_1, \check{\mathsf{ct}}_1), \mathsf{ct}_2 = (\hat{\mathsf{ct}}_2, \check{\mathsf{ct}}_2) \in R^2_{Q_\ell} \times R^2_{Q_\ell}$ as follows:

$$(\hat{\mathsf{ct}}_1, \check{\mathsf{ct}}_1) \otimes^2 (\hat{\mathsf{ct}}_2, \check{\mathsf{ct}}_2) := (\hat{\mathsf{ct}}_1 \otimes \hat{\mathsf{ct}}_2, \hat{\mathsf{ct}}_1 \otimes \check{\mathsf{ct}}_2 + \hat{\mathsf{ct}}_2 \otimes \check{\mathsf{ct}}_1).$$

The Tensor operation is crucial for reducing modulus consumption, as it consumes modulus 2^k instead of 2^{2k} by discarding the product $\check{\mathsf{ct}}_1 \otimes \check{\mathsf{ct}}_2$ of the ciphertexts components corresponding to the two remainders. Note that discarding $\check{\mathsf{ct}}_1 \otimes \check{\mathsf{ct}}_2$ introduces a new source of numerical error for the underlying plaintext. Overall for Mult^2, this leads to the main new numerical error, of the order of $\|\Delta^{-1} \cdot \mathsf{Dec}(\check{\mathsf{ct}}_1 \otimes \check{\mathsf{ct}}_2)\|_\infty$ where Δ is the scaling factor for messages.[1] If this is less than the other error terms (for example due to rounding ciphertexts in the rescaling step), the impact of the numerical inaccuracy of Tensor^2 on the precision of a plaintext remains limited. A detailed error analysis is provided in Section 4.

Relinearization and rescaling. If we apply Relin and RS operations to both decomposed ciphertext components in parallel, the error added to the high part $\hat{\mathsf{ct}}$ ruins the accuracy of the underlying plaintext, exactly like in the decompose-then-encrypt failed attempt described above. Instead, we rely on a 'raising the modulus' strategy that is numerically much more advantageous (similarly to encrypt-then-decompose outperforming decompose-then-encrypt). Given a pair of ciphertexts $(\hat{\mathsf{ct}}, \check{\mathsf{ct}}) \in R^2_{Q_\ell} \times R^2_{Q_\ell}$, we define the Relin^2 and

[1] Recall that in CKKS, a real number to be encrypted is first multiplied by a scaling factor before being rounded to an integer; the scaling factor corresponds to the precision of the encoding.

RS^2 operations as follows:

$$\mathsf{Relin}^2(\hat{\mathsf{ct}}, \check{\mathsf{ct}}) := \mathsf{DCP}_{2^k}(\mathsf{Relin}(2^k \cdot \hat{\mathsf{ct}})) + (0, \mathsf{Relin}(\check{\mathsf{ct}})),$$
$$\mathsf{RS}^2(\hat{\mathsf{ct}}, \check{\mathsf{ct}}) := (\mathsf{RS}(\hat{\mathsf{ct}}), \mathsf{RS}(2^k \cdot \hat{\mathsf{ct}} + \check{\mathsf{ct}}) - 2^k \cdot \mathsf{RS}(\hat{\mathsf{ct}})).$$

These operations may introduce errors on the plaintexts underlying the high parts, but they are compensated by matching terms in the low parts. A detailed analysis is presented in Section 4.1. Finally, we define Mult^2 as $\mathsf{RS}^2 \circ \mathsf{Relin}^2 \circ \mathsf{Tensor}^2$.

Growth of the low part. When a ciphertext ct is first decomposed, all coefficients of its low part $\check{\mathsf{ct}}$ are bounded as $2^k/2$. Therefore, its decryption $\mathsf{Dec}(\check{\mathsf{ct}})$ is also small. However, as we proceed with homomorphic computations and in particular multiplications, the plaintext underlying the low part grows, mostly due to the Tensor^2 tensor operation. In turn, the larger the low part, the greater the error that occurs during the Tensor^2 tensor operation and hence during the Mult^2 multiplication. This effect amplifies when Mult^2 is applied many times.

To mitigate this phenomenon, we propose to *recombine and decompose* the current ciphertext to make the decryption of its low part as small as it was originally, and hence prevent further error growth. A use of a large divisor 2^k saves more on the modulus consumption front, but increases the error caused from tensoring, which in turn requires an earlier recombine-and-decompose step. Therefore, *an appropriate divisor* should be selected for the application under scope. For an example parameter set with $N = 2^{15}$, our experiments in Figure 1 illustrate that recombination is interesting after 6 sequential applications of Mult^2, with a divisor of 23 bits.

1.3 Double-CKKS and Tuple-CKKS

Multiplication on decomposed ciphertexts can be completed with decomposed versions of the other basic homomorphic operations, resulting in a double-precision version of CKKS which we call Double-CKKS.

Our approach can be extended to decompose a ciphertext into several ciphertexts. We call the number of components the tuple length t. Double-CKKS corresponds to $t = 2$, and can save modulus consumption up to by half, with the same asymptotic cost.

As we use larger t, t-Tuple-CKKS can reduce the modulus consumption further, up to a factor $1/t$ of the initial size for very large precision. However, the number of pieces for a ciphertext becomes t so that the total size of t-Tuple CKKS ciphertexts remains the same. The number of integer operations is asymptotically increased by about t times for $t \leq o(\log N)$ (i.e., in a regime where the Number Theoretic Transforms dominate the overall cost), but it is similar to that of the original CKKS multiplication when taking into account that the modulus reduction leads to smaller integers for the integer operations.

1.4 Asymptotic gain

We compare the computational complexity of Mult and Mult^2. The dominating step in homomorphic multiplication is relinearization, which contains a lot of Number Theoretic Tranforms (NTT). The number of relinearizations is 2 for Mult^2 and 1 for usual Mult.

We now argue that the costs of these relinearizations differs. As Mult^2 has a modulus consumption that is essentially half of that of Mult, for the same computation, one can halve the modulus bit-sizes by relying on Mult^2 rather than Mult. Further, we can decrease the ring dimension by using the fact that the maximum modulus is halved. Since the maximum modulus bit-size available while satisfying certain security level is proportional to the ring dimension, one may halve the ring dimension while maintaining the same security.

Overall, by taking into account the increase of number of relinearizations and decreases in modulus bit-sizes and ring dimensions, we obtain that Mult^2 should enable a decrease by a factor 2 of latency and ciphertext size, while keeping throughput essentially identical. We stress that these estimates are quite approximate as they neglect lower-order terms.

1.5 Concrete examples

We run experiments for concrete examples motivated by several interesting applications.

Increased homomorphic capacity. One basic application of Double-CKKS is to increase the number of sequential multiplications while keeping the same scheme parameters. Experimental results on this scenario can be found in Table 2. For ring degree $N = 2^{15}$, Double-CKKS enables a capacity of 18 sequential multiplications while the classical CKKS scheme allows only 13 sequential multiplications. The data is for similar precision and for largest moduli of similar sizes maximized to retain 128 bits of security.

Increased precision. Since Double-CKKS reduces the modulus consumption during multiplication, we can get high-precision homomorphic encryption with smaller parameters than before. For multiplication depth 8 and 100 bit scaling factor, CKKS needs a maximal modulus of at least 1,000 bits (including the key switching auxiliary modulus). To enjoy 128 bits of security, this leads to choosing a ring degree $N = 2^{16}$.

As illustrated in Table 3, Double-CKKS enables the same computation with degree $N = 2^{15}$ and a maximal modulus of only 680 bits. Furthermore, it decreases the multiplication latency from 270ms to 179ms, the maximal-level ciphertext size from 14.8MB to 5.08MB, and more remarkably the maximal-level switching key size from 74MB to 30.6MB. This is useful for many applications where achieving high precision is crucial. These include Multi-key HE [9, 14, 36] or Threshold FHE [4] and applications requiring $\text{IND-CPA}^{\text{D}}$ security [34, 35].

1.6 Related works

Saving consumption of ciphertext modulus has been a focus of various works. Gadget decomposition was for example introduced to avoid modulus consumption in key switching. It comes in several flavours, including bit decomposition [5, 6], digit decomposition [13] and Residue Number System (RNS) based decomposition [3, 20, 21]. However, gadget decomposition not only slows key switching by a factor equal to the gadget rank *dnum*, but also increases key size by the same factor. The SEAL library [40] uses the largest *dnum* possible as default: as a result, the size of each switching key[2] is 142.6MB for degree $N = 2^{15}$ and 851.4MB for

[2]The key size depends on the multiplicative depth and precision to be used, the numbers we provide are for the generally available parameter sets.

degree $N = 2^{16}$. The Gentry-Sahai-Waters scheme [19], relying on bit decomposition, drew attention due to its small noise growth. In addition, several works focused on reducing the modulus consumption in specific operations including linear transformation and bootstrapping [2, 7, 8, 21, 24, 31–33]. However, these studies focused on massive computations such as bootstrapping, and did not improve homomorphic multiplication itself.

One of the important advantages of modulus saving is that ring dimension can be reduced. Our Mult^2 algorithm roughly halves the modulus consumption in bits, allowing in turn to halve the ring dimension. An alternative way to decrease the ring dimension is to use ciphertexts in module-LWE formats, as proposed in [6] and more recently studied in [39] in the context of CKKS. We may first compare rank-2 module-CKKS and double-CKKS. Module-CKKS keeps the same modulus as CKKS and a ciphertext has 3 ring elements instead of 2 for CKKS, while double-CKKS allows to halve bit-size of the maximum modulus and a ciphertext consists of 4 ring elements. As a result, a double-CKKS ciphertext is 33% smaller in bit-size than a rank-2 module-CKKS ciphertext. The number of relinearizations for double-CKKS and module-CKKS multiplications are 2 and 3 respectively, meaning that double-CKKS also wins in computation cost. The comparison becomes even more in favor of tuple-CKKS when a larger t is used: the number of relinearizations grows linearly versus quadratically, and there is a larger gap in modulus bit-sizes.

2 PRELIMINARIES

In this section, we briefly recall the CKKS scheme and its bootstrapping.

Notation. We use lower-case bold face for vectors. For a real number r, we let $\lceil r \rfloor$ denote its rounding to the nearest integer (downwards in case of a tie). Modular reduction by q is denoted by $[\cdot]_q$. When mapping the result to the integers, we take a representative in $(-q/2, q/2]$. We let $\langle \cdot, \cdot \rangle$ or simply $\cdot$ denote the inner product of two vectors and $\| \cdot \|_\infty$ the infinite norm. We use $x \leftarrow D$ to denote the sampling x according to distribution D. When a set S is used instead of a distribution, $x \leftarrow S$ means that x is sampled uniformly in S. Let $R = \mathbb{Z}[X]/(X^N + 1)$ be the ring of integers of the $2N$-th cyclotomic field with a power-of-two degree N. For an integer $q \geq 2$, we write $R_q = R/qR$. When clear from context, a polynomial $a(X)$ can be denoted by a, i.e., omitting the variable X.

Residue number system (RNS). Let $\mathcal{B} = \{q_0, \cdots, q_\ell\}$ be a set of primes and $Q_\ell = \prod_{i=0}^{\ell} q_i$. We use the notation $[\cdot]_\mathcal{B}$ to refer to the mapping from $\mathbb{Z}_{Q_\ell}$ to $\mathbb{Z}_{q_0} \times \ldots \times \mathbb{Z}_{q_\ell}$, where a is mapped to $[a]_\mathcal{B} = ([a]_{q_i})_{0 \leq i \leq \ell}$. By the Chinese Remainder Theorem (CRT), this mapping is a ring isomorphism. The representation $[a]_\mathcal{B}$ is known as the RNS representation of $a \in \mathbb{Z}_{Q_\ell}$. It enables component-wise arithmetic operations within the smaller rings $\mathbb{Z}_{q_i}$, which reduces computation costs.

2.1 The CKKS encryption scheme

In the CKKS scheme, messages are complex vectors while ciphertexts are elements in $R_Q \times R_Q$ for some integer Q. The modulus Q is typically a product of RNS integers and, for the sake of efficiency, arithmetic modulo Q is performed in the RNS form whenever possible. The canonical embedding $\mathsf{can} : \mathbb{R}[X]/(X^N + 1) \rightarrow \mathbb{C}^{N/2}$ maps $m(X) \in R$ into $\mathbf{m} \in \mathbb{C}^{N/2}$, by evaluating $m(X)$ at the primitive $2N$-th roots of unity $\xi_j = \xi^{5^j}$ for $0 \leq j < N/2$. We discard the subsequent ξ^{5^j}'s as they are conjugates of the first ones and do not carry any extra information: as a result, the map can is an isomorphism. We use can^{-1} to convert (encode) messages $\mathbf{m}$ and plaintexts $m(X)$.

Encoding and decoding. Beyond can, encoding and decoding for the CKKS scheme rely on a real number $\Delta > 0$ called the scaling factor.[3]

- **Encoding**: $m(X) \leftarrow \mathsf{Ecd}(\mathbf{m}, \Delta)$. Given a message $\mathbf{m} \in \mathbb{C}^{N/2}$ and the scaling factor Δ, the encoding map returns $m(X) = \lceil \Delta \cdot \mathsf{can}^{-1}(\mathbf{m}) \rfloor$.
- **Decoding**: $\mathbf{m} \leftarrow \mathsf{Dcd}(m(X), \Delta)$. Given a plaintext $m(X) \in R$ and the scaling factor Δ, the decoding map returns $\mathbf{m} = \mathsf{can}(\Delta^{-1} \cdot m(X))$.

Basic operations. We now recall the algorithms of the CKKS encryption scheme. Homomorphic multiplication and bootstrapping are described subsequently.

- **Setup**: params $\leftarrow \mathsf{FHE.Setup}(1^\lambda)$. Given the security parameter λ, return a degree N, a scaling factor Δ, a secret key Hamming weight h, a chain of moduli $Q_0 < \cdots < Q_L$, an auxiliary modulus P, two distributions χ_{enc} and χ_{err} over R and a decryption bound B_{Dec}.
- **Key generation**: $(\mathsf{sk}, \mathsf{pk}, \mathsf{swk}) \leftarrow \mathsf{KeyGen}(\mathsf{params})$. Return a secret key, public key and switching keys (including the relinearization key and rotation keys).
  - Sample $s \in R$ with coefficients in $\{-1, 0, 1\}$ and Hamming weight h, from a prescribed distribution, and return $\mathsf{sk} = (1, s)$.
  - Sample $a \leftarrow R_{Q_L}$ and $e \leftarrow \chi_{err}$; return $\mathsf{pk} = (b = [-a \cdot s + e]_{Q_L}, a) \in R_{Q_L}^2$.
  - Sample $a \leftarrow R_{PQ_L}$ and $e \leftarrow \chi_{err}$; return $\mathsf{swk} = (b = [-a \cdot s + e + P \cdot s']_{PQ_L}, a)$ where $s' \in R$ is a switching key. We consider $s' = s^2$ to obtain the relinearization key rlk and $s' = s(X^{5^j})$ for $1 \leq j \leq N/2 - 1$ to obtain the rotation key rk_j.
- **Encryption**: $\mathsf{ct} \leftarrow \mathsf{Enc}(m(X))$. Given a plaintext $m(X) \in R$, sample $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$; return $\mathsf{ct} = [v \cdot \mathsf{pk} + (m(X) + e_0, e_1)]_{Q_L}$.
- **Decryption**: $m(X) \leftarrow \mathsf{Dec}(\mathsf{ct})$. Given a ciphertext $\mathsf{ct} \in R_{Q_\ell}^2$ for some $0 \leq \ell \leq L$, return $m(X) = [\langle \mathsf{ct}, \mathsf{sk} \rangle]_{Q_\ell}$ if the latter has infinity norm $\leq B_{\mathsf{Dec}}$, else return an error symbol.
- **Rescale**: $\mathsf{ct}_{rs} \leftarrow \mathsf{RS}_q(\mathsf{ct})$. This operation requires q to be a factor of the ciphertext modulus Q_ℓ. Given a ciphertext $\mathsf{ct} \in R_{Q_\ell}^2$, return $ct_{rs} = \lceil q^{-1} \mathsf{ct} \rfloor \bmod (Q_\ell / q) \in R_{Q_\ell/q}^2$. If $h + 1 < Q_\ell / q$, then we have $\|\mathsf{RS}_q(\mathsf{ct}) \cdot \mathsf{sk} - q^{-1} \mathsf{ct} \cdot \mathsf{sk}\|_\infty \leq (h+1)/2$. We assume that h is set so that the condition holds every time we use RS in this work.

[3] To minimize the numerical errors, it is useful to consider a different scaling factor for each multiplication level of the circuit to be homomorphically evaluated; for the sake of simplicity, we do not consider this optimization.

- **Addition/subtraction**: $\mathsf{ct}_{add}/\mathsf{ct}_{sub} \leftarrow \mathsf{Add}/\mathsf{Sub}(\mathsf{ct}, \mathsf{ct}')$. Given two ciphertexts $\mathsf{ct}, \mathsf{ct}' \in R^2_{Q_\ell}$, return $\mathsf{ct}_{add} = [\mathsf{ct} + \mathsf{ct}']_{Q_\ell}$ (resp. $\mathsf{ct}_{sub} = [\mathsf{ct} - \mathsf{ct}']_{Q_\ell}$). Note that $\mathsf{Dec}(\mathsf{ct}_{add}) = \mathsf{Dec}(\mathsf{ct}) + \mathsf{Dec}(\mathsf{ct}')$ (resp. $\mathsf{Dec}(\mathsf{ct}_{sub}) = \mathsf{Dec}(\mathsf{ct}) - \mathsf{Dec}(\mathsf{ct}')$).

The scheme parameters are set so that after homomorphic evaluation of a circuit on freshly generated ciphertexts, the underlying plaintexts remain small and decrypt to a value that is close to the evaluation of the considered circuit on the input plaintexts. This is controlled by the B_{Dec} bound. Alternatively, we could define decryption as the inner product of the ciphertext with sk modulo Q_0, and prove in the analysis that the result is indeed small and correct. The first formulation is syntactically more convenient for presenting our contributions.

2.2 CKKS multiplication

We now focus on homomorphic multiplication which is our target operation. CKKS multiplication has the following signature.

- **Multiplication**: $\mathsf{ct}_{mult} \leftarrow \mathsf{Mult}(\mathsf{ct}, \mathsf{ct}')$. Given two ciphertexts $\mathsf{ct}, \mathsf{ct}' \in R_{Q_\ell} \times R_{Q_\ell}$ for Q_ℓ in the modulus chain that decrypt to m and m', return a ciphertext $\mathsf{ct}_{mult} \in R_{Q_{\ell-1}} \times R_{Q_{\ell-1}}$ that decrypts to $\approx m \cdot m'/\Delta$.

Note that the output ciphertext is over a modulus that is lower than the input ciphertexts. This motivates the use of a chain of moduli $Q_0 < \cdots < Q_L$, starting computations with ciphertexts defined modulo Q_L and progressively going down the modulus chain when multiplications are performed. Modulus consumption has a drastic impact on performance, as homomorphic evaluations with large multiplicative depth require a high modulus to start with, which leads to heavier run-times. When adding, subtracting or multiplying ciphertexts defined modulo Q_i and Q_j for $i < j$, one can use the rescale operation RS defined above to equalize the moduli to Q_i. Oppositely, to reduce modulus consumption, it is sometimes interesting to multiply $\mathsf{ct} \in R^2_{Q_i}$ by Q_j/Q_i to go to the larger modulus Q_j. This requires care as it also increases the underlying plaintext. For example, we use this approach in Definition 4.3 to avoid modulus consumption.

CKKS multiplication contains three steps: Tensor, Relin and RS.

- **Tensor**: $\mathsf{ct}_{ten} \leftarrow \mathsf{ct} \otimes \mathsf{ct}'$ or $\mathsf{ct}_{ten} \leftarrow \mathsf{Tensor}(\mathsf{ct}, \mathsf{ct}')$. Given two ciphertexts $\mathsf{ct} = (b, a)$ and $\mathsf{ct}' = (b', a') \in R^2_{Q_\ell}$, return $\mathsf{ct}_{ten} = (b \cdot b', -a \cdot b' - a' \cdot b, a \cdot a') \in R^3_{Q_\ell}$. Writing $\mathsf{sk} = (1, s)$, we have:
$$\mathsf{ct}_{ten} \cdot (1, s, s^2) = \mathsf{Dec}(\mathsf{ct}) \cdot \mathsf{Dec}(\mathsf{ct}').$$
- **Relinearize**: $\mathsf{ct}_{relin} \leftarrow \mathsf{Relin}(\mathsf{ct}_{ten}, \mathsf{rlk})$. Given a ciphertext $\mathsf{ct}_{ten} = (\mathsf{ct}, \mathsf{ct}', \mathsf{ct}'') \in R^3_{Q_\ell}$ and a relinearization key $\mathsf{rlk} \in R^2_{PQ_\ell}$, return $\mathsf{ct}_{relin} \in R^2_{Q_\ell}$ defined as follows:[4]
$$\mathsf{ct}_{relin} = \left(\mathsf{ct}, \mathsf{ct}'\right) + \left\lceil \frac{\mathsf{ct}'' \cdot \mathsf{rlk}}{P} \right\rfloor.$$
Note that Relin maps a ciphertext that decrypts under the extended key $(1, s, s^2)$ to a ciphertext that decrypts to a nearby plaintext under the secret key $\mathsf{sk} = (1, s)$. Indeed, it may be checked that there exists a (small) bound E_{Relin} such that $\|[(\mathsf{ct}_{relin} \cdot (1, s)) - \mathsf{ct}_{ten} \cdot (1, s, s^2)]_{Q_\ell}\|_\infty \le E_{\mathsf{Relin}}$.
- **Rescale**: $\mathsf{ct}_{rs} \leftarrow \mathsf{RS}_{q_\ell}(\mathsf{ct}_{relin})$ where $q_\ell = Q_\ell/Q_{\ell-1}$. The output ct_{rs} is an element of $R^2_{Q_{\ell-1}}$. The purpose of RS in homomorphic multiplication is to reduce the error and to maintain the scale factor. Indeed, both of them grow quadratically with the above operations: first, if both $\mathsf{Dec}(\mathsf{ct})$ and $\mathsf{Dec}(\mathsf{ct}')$ contain errors, then their product contains an error term that is the product of the errors; second, if both $\mathsf{Dec}(\mathsf{ct})$ and $\mathsf{Dec}(\mathsf{ct}')$ need to be divided by Δ when decoding, then their product needs to be divided by Δ^2 to achieve a homomorphic multiplication on plaintexts. Note that in the homomorphic multiplication algorithm, only the RS operation consumes ciphertext modulus.

Homomorphic multiplication of ciphertexts over modulus Q_ℓ is then defined as $\mathsf{Mult} := \mathsf{RS}_{q_\ell} \circ \mathsf{Relin} \circ \mathsf{Tensor}$. It may be showed that if $\mathsf{Dec}(\mathsf{ct})$ and $\mathsf{Dec}(\mathsf{ct}')$ have sufficiently small infinity norms compared to B_{Dec}, then
$$\left\| \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Mult}(\mathsf{ct}, \mathsf{ct}')) - \frac{\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}) \cdot \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}')}{\Delta} \right\|_\infty \le E_{\mathsf{Mult}},$$
for some (small) bound E_{Mult}.

2.3 CKKS bootstrapping

As homomorphic multiplications consume modulus, at some stage, one cannot perform homomorphic multiplications anymore. The bootstrapping algorithm increases the ciphertext modulus while decrypting to the same message (up to some small noise). CKKS bootstrapping consists of four steps: StC, ModRaise, CtS, and EvalMod.

StC and CtS are homomorphic evaluations of the discrete Fourier transform and its inverse, respectively. Both consist of several multiplications by constants and rotations. Rotation is performed using the rotation keys rk_j for some j and allows to map a ciphertext ct to another one that decrypts to $\approx \mathsf{Dec}(\mathsf{ct})(X^{5^j})$. ModRaise consists in viewing a ciphertext $\mathsf{ct} \in R_{Q_0} \times R_{Q_0}$ by Q_L/Q_0 and as a ciphertext in $R_{Q_L} \times R_{Q_L}$ that decrypts to $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}) + Q_0 \cdot I$ for some small integer I. EvalMod is the homomorphic evaluation of a polynomial approximation of the modular reduction function, used to remove the term $Q_0 \cdot I$ caused by ModRaise.

3 HOMOMORPHIC EUCLIDEAN DIVISION

When constructing multi-precision arithmetic using single-precision arithmetic, one decomposes a number into several pieces and define operations on them. In order to define an analogous system for the CKKS scheme, we first exhibit a homomorphic Euclidean division and use it to decompose ciphertexts into several pieces.

3.1 Homomorphic Euclidean division

Let $m \in R$ be a plaintext polynomial. By coefficient-wise extension of the Euclidean division of integers to polynomials, we define the Euclidean division of m by q as
$$m = \frac{m - [m]_q}{q} \cdot q + [m]_q.$$

[4] There exist variants, but the specific choice is irrelevant for describing our contributions.

The quotient is $(m - [m]_q)/q$, and the remainder is $[m]_q$. We define division on ciphertexts in the same way, by applying it on both ciphertext components.

Definition 3.1 (Ciphertext Euclidean division). *Let $q|Q$ be two integers. Let $\mathsf{ct} = (b, a) \in R_Q^2$ be a ciphertext. The remainder of ct by q is defined as*

$$\mathsf{Rem}_q(\mathsf{ct}) = ([b]_q, [a]_q) \in R^2.$$

The quotient of ct by q is defined as

$$\mathsf{Quo}_q(\mathsf{ct}) = \mathsf{RS}_q(\mathsf{ct}) = \frac{\mathsf{ct} - \mathsf{Rem}_q(\mathsf{ct})}{q} \ \in \ R_{Q/q}^2.$$

In the definition $\mathsf{Quo}_q(\mathsf{ct})$, the numerator is computed modulo Q, i.e., the remainder $\mathsf{Rem}_q(\mathsf{ct}) \in R^2$ is interpreted as an element of R_Q^2. As a result, the quotient $\mathsf{Quo}_q(\mathsf{ct})$ belongs to the ciphertext space of modulus Q/q. The remainder $\mathsf{Rem}_q(\mathsf{ct})$ is itself defined over $R \times R$ with coefficients in $(-q/2, q/2]$, but we will later view it as a ciphertext modulo Q/q.

Theorem 3.2. *Let $q|Q$ be two integers. Let $\mathsf{ct} \in R_Q^2$ and $\mathsf{sk} = (1, s)$ be a secret key with s of Hamming weight h. Let $m = [\mathsf{ct} \cdot \mathsf{sk}]_Q \in R$ and write $m = \hat{m} \cdot q + \check{m}$ with $\check{m} = [m]_q \in R$ and $\hat{m} = (m - [m]_q)/q \in R$. We have,*

$$\begin{aligned} \mathsf{Quo}_q(\mathsf{ct}) \cdot \mathsf{sk} &= \hat{m} + I \mod Q/q, \\ \mathsf{Rem}_q(\mathsf{ct}) \cdot \mathsf{sk} &= \check{m} - qI, \end{aligned}$$

for some $I \in R$ satisfying $\|I\|_\infty \le (h+2)/2$.

Proof. Since $\mathsf{Quo}_q(\mathsf{ct})$ is an element of $R_{Q/q}^2$, the quantity $q \cdot \mathsf{Quo}_q(\mathsf{ct})$ can be viewed as an element of R_Q^2. Hence the definition of Quo gives the identity

$$\mathsf{ct} = q \cdot \mathsf{Quo}_q(\mathsf{ct}) + \mathsf{Rem}_q(\mathsf{ct}) \mod Q.$$

By taking the inner product with sk, we obtain:

$$m = q \cdot \left[\mathsf{Quo}_q(\mathsf{ct}) \cdot \mathsf{sk}\right]_{Q/q} + \mathsf{Rem}_q(\mathsf{ct}) \cdot \mathsf{sk} \mod Q.$$

Let $I = [\mathsf{Quo}_q(\mathsf{ct}) \cdot \mathsf{sk} - \hat{m}]_{Q/q} \in R$. By using the identity $m = \hat{m} \cdot q + \check{m}$, we observe that

$$\mathsf{Rem}_q(\mathsf{ct}) \cdot \mathsf{sk} = \check{m} - qI \mod Q.$$

We now show that I is small. Since $[\cdot]_q$ is a signed modular reduction, we have $\|[\mathsf{Rem}(\mathsf{ct})]_q\|_\infty \le q/2$. Thus

$$\|\mathsf{Rem}_q(\mathsf{ct}) \cdot \mathsf{sk}\|_\infty = \|\mathsf{Rem}_q(\mathsf{ct}) \cdot (1, s)\|_\infty \le (h+1)\frac{q}{2}.$$

As $\|\check{m}\|_\infty \le q/2$, we obtain that $\|I\|_\infty \le (h+2)/2$. □

3.2 Pair representation

We introduce a novel ciphertext representation, as a quotient-remainder pair rather than a single element of R_Q^2 for some Q.

Definition 3.3. *Let Q_ℓ be an element of the modulus chain (see Section 2.1) and $q_{\mathsf{div}} \ge 2$. Let $Q'_\ell = Q_\ell \cdot q_{\mathsf{div}}$. Let $\mathsf{ct} \in R_{Q'_\ell}^2$. The decomposition of ct is*

$$\mathsf{DCP}_{q_{\mathsf{div}}}(\mathsf{ct}) = \left(\mathsf{Quo}_{q_{\mathsf{div}}}(\mathsf{ct}), \mathsf{Rem}_{q_{\mathsf{div}}}(\mathsf{ct})\right) \in R_{Q_\ell}^2 \times R_{Q_\ell}^2.$$

Conversely, the recombination of $(\hat{\mathsf{ct}}, \check{\mathsf{ct}}) \in R_{Q_\ell}^2 \times R_{Q_\ell}^2$ is

$$\mathsf{RCB}_{q_{\mathsf{div}}}(\hat{\mathsf{ct}}, \check{\mathsf{ct}}) = q_{\mathsf{div}} \cdot \hat{\mathsf{ct}} + \check{\mathsf{ct}} \in R_{Q_\ell}^2.$$

An element of $R_Q^2 \times R_Q^2$ corresponds to two CKKS ciphertexts, which is why we refer to this decomposition as **Pair Representation**. Note that for any $\mathsf{ct} \in R_{Q'_\ell}^2$, we have

$$\mathsf{RCB}_{q_{\mathsf{div}}} \circ \mathsf{DCP}_{q_{\mathsf{div}}}(\mathsf{ct}) = [\mathsf{ct}]_{Q'_\ell}$$

over R (i.e., when casting the output of DCP to R^2). We emphasize that in general, the output of $\mathsf{RCB}_{q_{\mathsf{div}}}$ is only defined over $R_{Q_\ell}^2$, and has no reason to be well-defined modulo Q'_ℓ after homomorphic operations as described in Section 4 have been performed.

Theorem 3.2 states that applying $\mathsf{DCP}_{q_{\mathsf{div}}}$ on a ciphertext essentially performs Euclidean division on the underlying plaintexts. The following lemma states that applying $\mathsf{RCB}_{q_{\mathsf{div}}}$ on a pair of ciphertexts recombines the underlying plaintexts in base q_{div}.

Lemma 3.4. *For $(\hat{\mathsf{ct}}, \check{\mathsf{ct}}) \in R_{Q_\ell}^2 \times R_{Q_\ell}^2$ and a secret key $\mathsf{sk} \in R^2$, we have, modulo Q_ℓ:*

$$\mathsf{RCB}_{q_{\mathsf{div}}}(\hat{\mathsf{ct}}, \check{\mathsf{ct}}) \cdot \mathsf{sk} = (\hat{\mathsf{ct}} \cdot \mathsf{sk}) \cdot q_{\mathsf{div}} + (\check{\mathsf{ct}} \cdot \mathsf{sk}).$$

3.3 Tuple representation

We now extend the ciphertext pair representation to tuples.

Definition 3.5. *Let Q_ℓ an element of the modulus chain (see Section 2.1), $q_{\mathsf{div}} \ge 2$ and $e \ge 1$. Let $Q'_\ell = Q_\ell \cdot q_{\mathsf{div}}^{e-1}$. Let $\mathsf{ct} \in R_{Q'_\ell}^2$. Define*

$$\begin{aligned} (\hat{ct}_0, \check{\mathsf{ct}}_0) &= \mathsf{DCP}_{q_{\mathsf{div}}^{e-1}}(\mathsf{ct}), \\ (\hat{ct}_i, \check{\mathsf{ct}}_i) &= \mathsf{DCP}_{q_{\mathsf{div}}^{e-i-1}}(\check{ct}_{i-1}) \ \text{ for } \ 1 \le i \le e-2. \end{aligned}$$

For convenience, define $\hat{ct}_{e-1} = \check{ct}_{e-2}$. The decomposition of ct is:

$$\mathsf{DCP}_{q_{\mathsf{div}}}^e(\mathsf{ct}) = \left(\hat{ct}_0, \ldots, \hat{ct}_{e-2}, \hat{ct}_{e-1}\right) \in \left(R_{Q_\ell}^2\right)^e.$$

Conversely, the recombination of $(\hat{\mathsf{ct}}_0, \ldots, \hat{\mathsf{ct}}_{e-1}) \in (R_{Q_\ell}^2)^e$ is

$$\begin{aligned} &\mathsf{RCB}_{q_{\mathsf{div}}}^e(\hat{\mathsf{ct}}_0, \ldots, \hat{\mathsf{ct}}_{e-2}, \hat{\mathsf{ct}}_{e-1}) \\ &\qquad = \hat{\mathsf{ct}}_0 \cdot q_{\mathsf{div}}^{e-1} + \ldots + \hat{\mathsf{ct}}_{e-2} \cdot q_{\mathsf{div}} + \hat{\mathsf{ct}}_{e-1} \in R_{Q_\ell}^2. \end{aligned}$$

An element of $(R_Q^2)^e$ corresponds to e CKKS ciphertexts, which is why we refer to this decomposition as ***e*-Tuple Representation**. We call t the tuple length. Pair representation corresponds to $t = 2$. Note that for any $\mathsf{ct} \in R_{Q'_\ell}^2$, we have

$$\mathsf{RCB}_{q_{\mathsf{div}}}^e \circ \mathsf{DCP}_{q_{\mathsf{div}}}^e(\mathsf{ct}) = [\mathsf{ct}]_{Q'_\ell}$$

over R (i.e., when casting the output of DCP to R^e).

4 HOMOMORPHIC DOUBLE-PRECISION MULTIPLICATION

In this section, we describe our novel method to increase multiplication precision by constructing a new multiplication consisting of low precision multiplications. Before introducing our algorithm, we recall a classical technique used to double precision in fixed-point arithmetic.

Let m be an integer. Decompose m into a pair of integers which correspond to quotient and remainder by 2^k where k is a positive integer:

$$\hat{m} = \mathsf{Quo}_{2^k}(m), \ \check{m} = \mathsf{Rem}_{2^k}(m), \ m = 2^k \cdot \hat{m} + \check{m},$$

with $|\check{m}| \leq 2^k/2$. Then, the multiplication of two decomposed integers $m_1 = 2^k \cdot \hat{m}_1 + \check{m}_1$ and $m_2 = 2^k \cdot \hat{m}_2 + \check{m}_2$ satisfies:

$$m_1 \cdot m_2 = 2^{2k} \cdot \hat{m}_1 \cdot \hat{m}_2 + 2^k \cdot (\hat{m}_1 \cdot \check{m}_2 + \hat{m}_2 \cdot \check{m}_1) + \check{m}_1 \cdot \check{m}_2.$$

For fixed-point computation with relative error $\approx 2^{-2k}$, the last component $\check{m}_1 \cdot \check{m}_2$ is not relevant. Therefore, we can define the multiplication as follows:

$$(\hat{m}_1, \check{m}_1) \times (\hat{m}_2, \check{m}_2) := (\hat{m}_1 \cdot \hat{m}_2, \hat{m}_1 \cdot \check{m}_2 + \hat{m}_2 \cdot \check{m}_1).$$

We have already covered how to homomorphically perform a Euclidean division in Section 3. In this section, we apply the technique above for homomorphic multi-precision multiplication.

Note that homomorphic multi-precision addition and subtraction can be performed componentwise on pair representations of ciphertexts, and that key switching is studied below as a the relinearization component of multiplication (see Definition 4.3). This provides a complete set of instructions for homomorphic double-precision arithmetic. All components of bootstrapping are hence enabled for pair representations of ciphertexts, with the exception of modulus raising, for which we propose to perform $\mathsf{DCP} \circ \mathsf{ModRaise} \circ \mathsf{RCB}$.

4.1 Tools

As seen in Section 2.2, CKKS ciphertext multiplication consists of Tensor, Relin and RS operations. We define the corresponding operations for pair representations of ciphertexts.

Definition 4.1 (Pair tensor). *Let* $\mathsf{CT}_1 = (\hat{\mathsf{ct}}_1, \check{\mathsf{ct}}_1), \mathsf{CT}_2 = (\hat{\mathsf{ct}}_2, \check{\mathsf{ct}}_2) \in R^2_{Q_\ell} \times R^2_{Q_\ell}$ *be ciphertext pairs. The tensor of* CT_1 *and* CT_2 *is defined as*

$$\mathsf{CT}_1 \otimes^2 \mathsf{CT}_2 = \left(\hat{\mathsf{ct}}_1 \otimes \hat{\mathsf{ct}}_2, \hat{\mathsf{ct}}_1 \otimes \check{\mathsf{ct}}_2 + \check{\mathsf{ct}}_1 \otimes \hat{\mathsf{ct}}_2\right) \in R^3_{Q_\ell} \times R^3_{Q_\ell}.$$

We will also use the notation $\mathsf{Tensor}^2(\mathsf{CT}_1, \mathsf{CT}_2)$.

The Tensor^2 operation discards the component $\check{\mathsf{ct}}_1 \otimes \check{\mathsf{ct}}_2$, unlike the Tensor operation from Section 2.2. The following lemma formalizes the relationship between Tensor^2 and Tensor.

Lemma 4.2. *Let* $\mathsf{CT}_i = (\hat{\mathsf{ct}}_i, \check{\mathsf{ct}}_i) \in R^2_{Q_\ell} \times R^2_{Q_\ell}$ *be a ciphertext pair and* $\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}_i) = \mathsf{ct}_i$ *for* $i \in \{1, 2\}$. *Let* $\mathsf{sk} = (1, s) \in R^2$ *be a secret key. Then, modulo* Q_ℓ*:*

$$(\mathsf{ct}_1 \otimes \mathsf{ct}_2) \cdot (1, s, s^2) = q_{\mathsf{div}} \cdot (\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}_1 \otimes^2 \mathsf{CT}_2)) \cdot (1, s, s^2) + (\check{\mathsf{ct}}_1 \cdot \mathsf{sk}) \cdot (\check{\mathsf{ct}}_2 \cdot \mathsf{sk}).$$

Now, assume that $\|\mathsf{Dec}(\hat{\mathsf{ct}}_i)\|_\infty \leq \hat{M}$ *and* $\|\mathsf{Dec}(\check{\mathsf{ct}}_i)\|_\infty \leq \check{M}$ *for all* $i \in \{1, 2\}$ *and for some* $\hat{M}, \check{M}$ *satisfying* $N(\hat{M} q_{\mathsf{div}} + \check{M})^2 < Q_\ell/2$. *Then we have:*

$$\left\| [(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}_1 \otimes^2 \mathsf{CT}_2)) \cdot (1, s, s^2)]_{Q_\ell} - \frac{1}{q_{\mathsf{div}}} [(\mathsf{ct}_1 \otimes \mathsf{ct}_2) \cdot (1, s, s^2)]_{Q_\ell} \right\|_\infty \leq \frac{N \check{M}^2}{q_{\mathsf{div}}}.$$

Proof. Let $\hat{m}_i = \mathsf{Dec}_{\mathsf{sk}}(\hat{\mathsf{ct}}_i)$ and $\check{m}_i = \mathsf{Dec}_{\mathsf{sk}}(\check{\mathsf{ct}}_i)$ for $i \in \{1, 2\}$. Since

$$\mathsf{RCB}(\mathsf{CT}_1 \otimes^2 \mathsf{CT}_2) = q_{\mathsf{div}} \cdot (\hat{\mathsf{ct}}_1 \otimes \hat{\mathsf{ct}}_2) + (\hat{ct}_1 \otimes \check{\mathsf{ct}}_2 + \check{\mathsf{ct}}_1 \otimes \hat{\mathsf{ct}}_2),$$

we have, modulo Q_ℓ,

$$(\mathsf{RCB}(\mathsf{CT}_1 \otimes^2 \mathsf{CT}_2)) \cdot (1, s, s^2) = q_{\mathsf{div}} \cdot (\hat{m}_1 \hat{m}_2) + (\hat{m}_1 \check{m}_2 + \check{m}_1 \hat{m}_2).$$

Meanwhile, the following also holds, modulo Q_ℓ:

$$\begin{aligned}(\mathsf{ct}_1 \otimes \mathsf{ct}_2) \cdot (1, s, s^2) &= \left((q_{\mathsf{div}} \cdot \hat{ct}_1 + \check{\mathsf{ct}}_1) \otimes (q_{\mathsf{div}} \cdot \hat{\mathsf{ct}}_2 + \check{\mathsf{ct}}_2)\right) \cdot (1, s, s^2) \\ &= q_{\mathsf{div}}^2 (\hat{m}_1 \hat{m}_2) + q_{\mathsf{div}} (\hat{m}_1 \check{m}_2 + \check{m}_1 \hat{m}_2) + (\check{m}_1 \check{m}_2).\end{aligned}$$

This gives the first part of the result.

The condition $N(\hat{M} q_{\mathsf{div}} + \check{M})^2 < Q_\ell/2$ ensures that both left and right hand sides of the equation in the lemma statement have infinity norms $< Q_\ell/2$, implying that the following holds over R:

$$\begin{aligned}[(\mathsf{ct}_1 \otimes \mathsf{ct}_2) \cdot (1, s, s^2)]_{Q_\ell} &= q_{\mathsf{div}} \cdot [(\mathsf{RCB}(\mathsf{CT}_1 \otimes^2 \mathsf{CT}_2)) \cdot (1, s, s^2)]_{Q_\ell} \\ &+ [\check{\mathsf{ct}}_1 \cdot \mathsf{sk}]_{Q_\ell} \cdot [\check{\mathsf{ct}}_2 \cdot \mathsf{sk}]_{Q_\ell}.\end{aligned}$$

The result follows from bounding $\|[\check{\mathsf{ct}}_1 \cdot \mathsf{sk}]_{Q_\ell} \cdot [\check{\mathsf{ct}}_2 \cdot \mathsf{sk}]_{Q_\ell}\|_\infty$. □

Lemma 4.2 states that if the underlying plaintexts are sufficiently small, then Tensor^2 applied to CT_1 and CT_2 decrypts to approximately the same plaintext as Tensor of ct_1 and ct_2 divided by q_{div}, over R. Note that this would not be ensured if we only had the first part of Lemma 4.2, as we would have a division by q_{div} modulo Q_ℓ. Importantly, the Tensor^2 operation somewhat contains a rescaling by q_{div}, but without modulus consumption (the modulus Q_ℓ remains the same). This contributes to reducing modulus consumption in multiplication.

The lemma could have been stated with a joint upper bound $M = \max(\hat{M}, \check{M})$. We make it more precise as, later, they will play asymmetric roles: the numerical errors will be directly impacted by $\check{M}$, whereas $\hat{M}$ is mostly involved in the correctness of computations.

Definition 4.3 (Pair relinearize). *Let* $\mathsf{CT} = (\hat{\mathsf{ct}}, \check{\mathsf{ct}}) \in R^3_{Q_\ell} \times R^3_{Q_\ell}$ *be an output of* Tensor_2. *The relinearization of* CT *is defined as*

$$\mathsf{Relin}^2(\mathsf{CT}) = \mathsf{DCP}_{q_{\mathsf{div}}}(\mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}})) + (0, \mathsf{Relin}(\check{\mathsf{ct}})).$$

Note that the same algorithm may be used for other instanciations of key switching such as rotations.

A naive approach to relinearize CT would be to relinearize each component independently, but this introduces a devastating numerical error to the left hand side component that ruins the plaintext computations. Instead, we raise the left hand side component and decompose it. Observe that

$$\begin{aligned}\mathsf{Relin}(\mathsf{RCB}_{q_{\mathsf{div}}}(\hat{\mathsf{ct}}, \check{\mathsf{ct}})) &= \mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}} + \check{\mathsf{ct}}) \\ &\approx \mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}}) + \mathsf{Relin}(\check{\mathsf{ct}}).\end{aligned}$$

This gives an approximate identity

$$\mathsf{DCP}_{q_{\mathsf{div}}} \circ \mathsf{Relin} \circ \mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}) \approx \mathsf{Relin}^2(\mathsf{CT}),$$

from which we derive the following correctness statement.

Lemma 4.4. *Let* $\mathsf{CT} \in R^3_{Q_\ell} \times R^3_{Q_\ell}$ *and* $\mathsf{sk} = (1, s) \in R^2$ *a secret key with* s *of Hamming weight* h. *Then the quantity*

$$\left[\left(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{Relin}^2(\mathsf{CT}))\right) \cdot (1, s) - \left(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT})\right) \cdot (1, s, s^2)\right]_{Q_\ell}$$

has infinity norm $\leq E_{\mathsf{Relin}} + h$. *Now, assume that* $\|[\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}) \cdot (1, s, s^2)]_{Q_\ell}\|_\infty \leq M$ *for some* M *satisfying* $2(M + E_{\mathsf{Relin}} + h) < Q_\ell/2$. *Then the quantity*

$$\left[\left(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{Relin}^2(\mathsf{CT}))\right) \cdot (1, s)\right]_{Q_\ell} - \left[\left(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT})\right) \cdot (1, s, s^2)\right]_{Q_\ell}$$

also has infinity norm $\leq E_{\mathsf{Relin}} + h$.

Proof. Write $\mathsf{CT} = (\hat{\mathsf{ct}}, \check{\mathsf{ct}})$. By linearity of RCB, we have, modulo Q_ℓ:

$$\mathsf{RCB}(\mathsf{Relin}^2(\mathsf{CT})) = \mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}}) + \mathsf{Relin}(\check{\mathsf{ct}}).$$

By using the definition of Relin, we obtain, modulo Q_ℓ, that:

$$\mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}}) + \mathsf{Relin}(\check{\mathsf{ct}}) = \mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}} + \check{\mathsf{ct}}) + (0, e),$$

for some $e \in R$ satisfying $\|e\|_\infty \leq 1$ (it is $\leq 3/2$ as there are three Relin roundings involved, but it must be integral). We then obtain, still modulo Q_ℓ:

$$\Big(\mathsf{RCB}(\mathsf{Relin}^2(\mathsf{CT}))\Big) \cdot (1, s) = \Big(\mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}} + \check{\mathsf{ct}})\Big) \cdot (1, s) + s \cdot e.$$

Note that $\|s \cdot e\|_\infty \leq h$. Using the correctness property of Relin (see Section 2.2), we have:

$$\left\| \left[\Big(\mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}} + \check{\mathsf{ct}})\Big) \cdot (1, s) - \Big(\hat{\mathsf{ct}} \cdot q_{\mathsf{div}} + \check{\mathsf{ct}}\Big) \cdot (1, s, s^2) \right]_{Q_\ell} \right\|_\infty \leq E_{\mathsf{Relin}}.$$

This gives the first part of the result. The condition $2(\hat{M} q_{\mathsf{div}} \check{M} + E_{\mathsf{Relin}} + h) < Q_\ell/2$ ensures that both terms $[(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{Relin}^2(\mathsf{CT}))) \cdot (1, s)]_{Q_\ell}$ and $[(\hat{\mathsf{ct}} \cdot q_{\mathsf{div}} + \check{\mathsf{ct}}) \cdot (1, s, s^2)]_{Q_\ell}$ have infinity norms $< Q_\ell/4$, which leads to the result. □

Another naive approach for relinearization would consist in relinearizing after recombining: $\mathsf{DCP}_{q_{\mathsf{div}}} \circ \mathsf{Relin} \circ \mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT})$. Note that this consumes modulus: both RCB and Relin keep the modulus Q_ℓ of CT, but DCP decreases the modulus (by a factor of q_{div}). Oppositely, our approach does not consume modulus. As $q_{\mathsf{div}} \cdot \hat{\mathsf{ct}}$ is defined modulo $q_{\mathsf{div}} \cdot Q_\ell$, so is $\mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}})$, and hence the first term of $\mathsf{Relin}^2(\mathsf{CT})$ is defined modulo Q_ℓ. Also, the second term computes $(\mathsf{Relin}(\check{\mathsf{ct}}))$ without consuming any modulus.

Definition 4.5 (Pair rescale). *Let* $\mathsf{CT} = (\hat{\mathsf{ct}}, \check{\mathsf{ct}}) \in R_{Q_\ell}^2 \times R_{Q_\ell}^2$ *be a ciphertext pair. Let* $q_\ell = Q_\ell / Q_{\ell-1}$. *The rescale of* CT *is defined as*

$$\mathsf{RS}_{q_\ell}^2(\mathsf{CT}) = \Big(\mathsf{RS}_{q_\ell}(\hat{\mathsf{ct}}),\ \mathsf{RS}_{q_\ell}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}} + \check{\mathsf{ct}}) - q_{\mathsf{div}} \cdot \mathsf{RS}_{q_\ell}(\hat{\mathsf{ct}})\Big).$$

It belongs to $R_{Q_{\ell-1}}^2 \times R_{Q_{\ell-1}}^2$.

Note that the following equality holds:

$$\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{RS}_{q_\ell}^2(\mathsf{CT})) = \mathsf{RS}_{q_\ell}(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT})).$$

To achieve the same, a naive approach to rescale CT would consist in rescaling after recombining: $\mathsf{DCP}_{q_{\mathsf{div}}} \circ \mathsf{RS}_{q_\ell} \circ \mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT})$. This consumes more modulus: a factor q_ℓ is lost due to RS_{q_ℓ} and a factor q_{div} is lost due to $\mathsf{DCP}_{q_{\mathsf{div}}}$. The function $\mathsf{RS}_{q_\ell}^2$ only consumes a factor q_ℓ.

Lemma 4.6. *Let* $\mathsf{CT} \in R_{Q_\ell}^2 \times R_{Q_\ell}^2$ *be a ciphertext pair. Let* $q_\ell = Q_\ell / Q_{\ell-1}$. *Let* $\mathsf{sk} = (1, s) \in R^2$ *be a secret key with* s *of Hamming weight* h. *Then the quantity*

$$\left[\Big(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{RS}_{q_\ell}^2(\mathsf{CT}))\Big) \cdot (1, s)\right]_{Q_{\ell-1}} - \frac{1}{q_\ell}\left[\Big(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT})\Big) \cdot (1, s)\right]_{Q_\ell}$$

has infinity norm $\leq (h+1)/2$.

Proof. We have, modulo $Q_{\ell-1}$,

$$\Big(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{RS}_{q_\ell}^2(\mathsf{CT}))\Big) \cdot (1, s) = \Big(\mathsf{RS}_{q_\ell}(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}))\Big) \cdot (1, s).$$

Now, to complete the proof, note that

$$\left[\Big(\mathsf{RS}_{q_\ell}(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}))\Big) \cdot (1, s)\right]_{Q_{\ell-1}} - \frac{1}{q_\ell}\left[\Big(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT})\Big) \cdot (1, s)\right]_{Q_\ell}$$

has infinity norm $\leq (h+1)/2$. □

4.2 Multiplication for pair representations

We are now ready to define pair multiplication.

Definition 4.7 (Pair multiply). *We define multiplication of ciphertext pairs over* Q_ℓ *as* $\mathsf{Mult}^2 := \mathsf{RS}_{q_\ell}^2 \circ \mathsf{Relin}^2 \circ \mathsf{Tensor}^2$, *where* $q_\ell = Q_\ell / Q_{\ell-1}$. *The result is a ciphertext pair over modulus* $Q_{\ell-1}$.

By combining Lemmas 4.2, 4.4 and 4.6, we obtain the following theorem.

Theorem 4.8. *Let* $\mathsf{CT}_1 = (\hat{\mathsf{ct}}_1, \check{\mathsf{ct}}_1), \mathsf{CT}_2 = (\hat{\mathsf{ct}}_2, \check{\mathsf{ct}}_2) \in R_{Q_\ell}^2 \times R_{Q_\ell}^2$ *be ciphertext pairs. Let* $q_\ell = Q_\ell / Q_{\ell-1}$ *and* $\mathsf{sk} = (1, s) \in R^2$ *be a secret key with* s *of Hamming weight* h. *Assume that* $\|\mathsf{Dec}(\hat{\mathsf{ct}}_i)\|_\infty \leq \hat{M}$ *and* $\|\mathsf{Dec}(\check{\mathsf{ct}}_i)\|_\infty \leq \check{M}$ *for all* $i \in \{1, 2\}$ *and for some* $\hat{M}, \check{M}$ *satisfying* $N(\hat{M} q_{\mathsf{div}} + \check{M})^2 + E_{\mathsf{Relin}} + h < Q_\ell/2$. *Then*

$$\left[\Big(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{Mult}^2(\mathsf{CT}_1, \mathsf{CT}_2))\Big) \cdot \mathsf{sk}\right]_{Q_{\ell-1}} - \frac{1}{q_\ell}\left[\Big(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}_1) \cdot \mathsf{sk}\Big) \cdot \Big(\mathsf{RCB}_{q_{\mathsf{div}}}(\mathsf{CT}_2) \cdot \mathsf{sk}\Big)\right]_{Q_\ell}$$

has infinity norm $\leq (N\check{M}^2/q_{\mathsf{div}} + E_{\mathsf{Relin}} + h)/q_\ell + (h+1)/2$.

Proof. The theorem condition on $\hat{M}$ and $\check{M}$ is more stringent than the one in Lemma 4.2, which we can hence apply. Let $\mathsf{CT} = (\hat{\mathsf{ct}}, \check{\mathsf{ct}})$ denote the output of Tensor^2. Using the proof of Lemma 4.2, we see that

$$\|\mathsf{RCB}(\mathsf{CT}) \cdot (1, s, s^2)\|_\infty \leq \frac{N(\hat{M} q_{\mathsf{div}} + \check{M})^2}{q_{\mathsf{div}}}.$$

One then observes that the theorem condition on $\hat{M}$ and $\check{M}$ implies the one in Lemma 4.4, which we can hence apply. Finally, applying Lemma 4.6 and collecting terms provides the result. □

The condition on $\hat{M}$ and $\check{M}$ is to ensure correctness of the computation, up to an error term whose main component is $N\check{M}^2/(q_{\mathsf{div}} q_\ell)$. This term does not appear in the classical CKKS homomorphic multiplication, as it stems from the dropping of the product of the low parts in the definition of Tensor^2. Due to its importance, the growth of the quantity $\check{M}$ should be carefully bounded through homomorphic computations involving sequential multiplications.

Modulus consumption. Note that the size of $\check{M}$ is roughly $C q_{\mathsf{div}}^2$ where $C > 1$ is a small quantity. The exact size will be discussed in the following subsection. Here we explain how to set the sizes of q_{div} and q_ℓ. In order to keep the main component $N\check{M}/(q_{\mathsf{div}} q_\ell) \approx NC^2 \cdot q_{\mathsf{div}}/q_\ell$ small, the prime q_ℓ should be at least as large as small multiple of q_{div}. We choose it minimal under this constraint. Since we divide the plaintext by q_{div} in Tensor^2 and rescale by q_ℓ in $\mathsf{RS}_{q_\ell}^2$, the overall scale is divided by $q_{\mathsf{div}} q_\ell$ after a multiplication. Hence we have a relation $\Delta \approx q_{\mathsf{div}} q_\ell$. Since we chose q_ℓ slightly larger than q_{div}, this implies that Δ is $\approx q_\ell^2$. We can observe the main advantage of Mult^2 over classical CKKS homomorphic multiplication that for the same modulus consumption we can handle plaintexts that have twice larger bit-sizes.

Efficiency. In terms of run-time, executing Mult^2 requires 3 calls to Tensor, 2 calls to Relin and 2 calls to RS. Note that only Relin and RS involve NTTs, so that the contribution of Tensor to the overall cost is negligible. Both Relin and RS have a costs that are quasi-linear in the bit-size of the working modulus. Comparatively, the classical Mult algorithm requires 1 call to Tensor, 1 call to Relin and 1 call to RS, which seems cheaper than Mult^2 at first sight. However, when using several multiplications sequentially for large plaintexts, Mult requires moduli that are twice larger due to its high modulus consumption. Overall, the costs of Mult and Mult_2 are then similar for performing a sequence of computations. In fact, as Mult_2 requires smaller moduli, one can also use smaller-degree rings for the same security, which gives it an advantage in terms of latency. In the case of fully homomorphic computations, bootstrapping dominates the cost, and Mult^2 is then much preferable as it enable bootstrapping with smaller parameters.

In terms of swk bit-size, Mult^2 outperforms Mult by a factor ≈ 2 due to the smaller moduli that it involves. As for run-time, this impact further increases if one takes into account that smaller moduli enable the use of smaller-degree rings.

4.3 Bounding the low parts

In Theorem 4.8, we observed that the infinity norm of the error of pair multiplications is bounded by a function of $\check{M}$, an upper bound of $\|\mathsf{Dec}(\check{\mathsf{ct}}_i)\|_\infty$ for $i \in \{1,2\}$. As the decryption of the low part $\|\mathsf{Dec}(\check{\mathsf{ct}})\|_\infty$ increases as we proceed with a sequence of multiplications, the multiplication error also grows. Since Theorem 3.2 gives the initial upper bound of the low part right after the first decomposition, it suffices to analyze the low part growth through a single multiplication (the analysis for addition and subtraction is elementary, and key switching is handled identically to the relinearization step of multiplication).

As described in [38], using the canonical embedding rather than the polynomial representation leads to tighter bounds on norm growth when evaluating a circuit. This observation was used in [38] for studying error terms, whereas we rely on it here to study the low parts. Recall that the canonical embedding $\mathsf{can} : \mathbb{R}[X]/(X^N+1) \to \mathbb{C}^{N/2}$ gives the relationship between messages and plaintexts (see Section 2.1). The main observation is that for two polynomials $m_1, m_2 \in R$, we have $\|\mathsf{can}(m_1 \cdot m_2)\|_\infty \le \|\mathsf{can}(m_1)\|_\infty \cdot \|\mathsf{can}(m_2)\|_\infty$, as opposed to $\|m_1 \cdot m_2\|_\infty \le N \cdot \|m_1\|_\infty \cdot \|m_2\|_\infty$. When considering multiple multiplications, the difference becomes a large power of N. Finally, note that for $m \in R$, we have $\|\mathsf{can}(m)\|_\infty / N \le \|m\|_\infty \le \|\mathsf{can}(m)\|_\infty$.

Theorem 4.9 (Growth of the low part). *Let* $\mathsf{CT}_1 = (\hat{\mathsf{ct}}_1, \check{\mathsf{ct}}_1)$ *and* $\mathsf{CT}_2 = (\hat{\mathsf{ct}}_2, \check{\mathsf{ct}}_2) \in R_{Q_\ell}^2 \times R_{Q_\ell}^2$ *be ciphertext pairs. Let* $q_\ell = Q_\ell / Q_{\ell-1} \ge 2$ *and* $\mathsf{sk} = (1, s) \in R^2$ *be a secret key with s of Hamming weight h. Suppose that* $\|\mathsf{can} \circ \mathsf{Dec}(\hat{\mathsf{ct}}_i)\|_\infty \le \hat{M}$ *and* $\|\mathsf{can} \circ \mathsf{Dec}(\check{\mathsf{ct}}_i)\|_\infty < \check{M}$ *for* $i \in \{1,2\}$ *and for some* $\hat{M}, \check{M}$. *Let* $\mathsf{CT}_{\mathsf{Mult}} = \mathsf{Mult}^2(\mathsf{CT}_1, \mathsf{CT}_2) = (\hat{\mathsf{ct}}_{\mathsf{Mult}}, \check{\mathsf{ct}}_{\mathsf{Mult}})$. *Then the following holds:*

$$\|\mathsf{can} \circ \mathsf{Dec}(\check{\mathsf{ct}}_{\mathsf{Mult}})\|_\infty \le \frac{2\hat{M}\check{M}}{q_\ell} + N\Big(\frac{E_{\mathsf{Relin}}}{q_\ell} + (h+3)(q_{\mathsf{div}}+1)\Big).$$

Proof. We analyze the growth for each step of $\mathsf{Mult}^2 = \mathsf{RS}^2_{q_\ell} \circ \mathsf{Relin}^2 \circ \mathsf{Tensor}^2$.

Define $\mathsf{CT}_{\mathsf{Tensor}} = \mathsf{Tensor}^2(\mathsf{CT}_1, \mathsf{CT}_2) = (\hat{\mathsf{ct}}_{\mathsf{Tensor}}, \check{\mathsf{ct}}_{\mathsf{Tensor}})$. By definition of Tensor_2, we have:

$$\left\|\mathsf{can}\left([\check{\mathsf{ct}}_{\mathsf{Tensor}} \cdot (1, s, s^2)]_{Q_\ell}\right)\right\|_\infty \le 2\hat{M}\check{M}.$$

Now, define $\mathsf{CT}_{\mathsf{Relin}} = \mathsf{Relin}^2(\mathsf{CT}_{\mathsf{Tensor}}) = (\hat{\mathsf{ct}}_{\mathsf{Relin}}, \check{\mathsf{ct}}_{\mathsf{Relin}})$. Note that $\check{\mathsf{ct}}_{\mathsf{Relin}}$ consists of two terms: the first one is an output of DCP whereas the second one is $\mathsf{Relin}(\check{\mathsf{ct}}_{\mathsf{Tensor}})$. By Theorem 3.2, the inner product of first term and sk has infinity norm $\le Nq_{\mathsf{div}}(h+3)/2$ (for the canonical embedding). Thanks to the bound above on $\check{\mathsf{ct}}_{\mathsf{Tensor}} \cdot (1, s, s^2)$ and the one quantifying the accuracy of Relin (see Section 2.2), the inner product of the second term with sk has infinity norm $\le NE_{\mathsf{Relin}} + 2\hat{M}\check{M}$ (for the canonical embedding). The triangle inequality then gives:

$$\left\|\mathsf{can}\left([\check{\mathsf{ct}}_{\mathsf{Relin}} \cdot (1, s)]_{Q_\ell}\right)\right\|_\infty \le 2\hat{M}\check{M} + N\Big(E_{\mathsf{Relin}} + q_{\mathsf{div}}\frac{h+3}{2}\Big).$$

Finally, we consider $\mathsf{CT}_{\mathsf{Mult}} = \mathsf{RS}^2(\mathsf{CT}_{\mathsf{Relin}})$. By definition of RS^2, we have:

$$\check{\mathsf{ct}}_{\mathsf{Mult}} = \frac{1}{q_\ell}\Big(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}}_{\mathsf{Relin}} + \check{\mathsf{ct}}_{\mathsf{Relin}}\Big) + \check{e} - q_{\mathsf{div}}\big(\frac{1}{q_\ell}\hat{\mathsf{ct}}_{\mathsf{Relin}} + \hat{e}\big),$$

for some $\hat{e}, \check{e}$ whose canonical embeddings have infinity norms $\le N/2$. This simplifies to

$$\check{\mathsf{ct}}_{\mathsf{Mult}} = \frac{1}{q_\ell}\check{\mathsf{ct}}_{\mathsf{Relin}} - q_{\mathsf{div}} \cdot \hat{e} + \check{e}.$$

By taking the inner product with sk, we obtain

$$\left\|\mathsf{can}\left([\check{\mathsf{ct}}_{\mathsf{Mult}} \cdot (1, s)]_{Q_{\ell-1}}\right)\right\|_\infty \le \frac{1}{q_\ell}\left\|\mathsf{can}\left([\check{\mathsf{ct}}_{\mathsf{Relin}} \cdot (1, s)]_{Q_\ell}\right)\right\|_\infty + \frac{N}{2}(h+1)(q_{\mathsf{div}}+1).$$

This leads to the claimed bound. □

The main term of the upper bound is $2\hat{M}\check{M}/q_\ell$, which is proportional to $\check{M}$. In the classical CKKS scheme, it is usually assumed that $\|\mathsf{Dcd} \circ \mathsf{Dec}(\mathsf{ct})\|_\infty \le 1$. Since $\mathsf{Dcd} = \Delta^{-1} \cdot \mathsf{can}$, this can be reinterpreted as $\|\mathsf{can} \circ \mathsf{Dec}(\mathsf{ct})\|_\infty \le \Delta$. For this reason, we can assume that $\hat{M} = \Delta / q_{\mathsf{div}}$. In this case, the main term is as large as $2\Delta\check{M}/(q_{\mathsf{div}} q_\ell) \approx 2\check{M}$, thanks to our choice of Δ at the end of Section 4.2. Therefore, the upper bound of the low part grows by ≈ 1 bit after each multiplication, if Δ is set properly.

5 HOMOMORPHIC MULTI-PRECISION MULTIPLICATION

In this Section, we extend our method to multiple precision. We generalize it with any tuple length $t \ge 2$. The proofs are omitted as they are direct adaptations of those of Section 4.

5.1 Tools

We define the operations corresponding to Tensor^2, Relin^2 and RS^2 for t-tuple representations of ciphertexts.

Definition 5.1 (Tuple tensor). *Let* $\mathsf{CT}_1 = (\hat{\mathsf{ct}}_{1,0}, \cdots, \hat{\mathsf{ct}}_{1,t-1})$, $\mathsf{CT}_2 = (\hat{\mathsf{ct}}_{2,0}, \cdots, \hat{\mathsf{ct}}_{2,t-1}) \in (R^2_{Q_\ell})^t$ *be ciphertext tuples. The tensor of* CT_1 *and* CT_2 *is defined as*

$$\hat{\mathsf{ct}}_{ten,i} = \sum_{j=0}^{i} \hat{\mathsf{ct}}_{1,j} \otimes \hat{\mathsf{ct}}_{2,i-j}, \text{ for all } i \in \{0, \cdots, t-1\}$$

$$\mathsf{CT}_1 \otimes^t \mathsf{CT}_2 = (\hat{\mathsf{ct}}_{ten,0}, \cdots, \hat{\mathsf{ct}}_{ten,t-1}) \in (R^3_{Q_\ell})^t.$$

We will also use the notation $\mathsf{Tensor}^t(\mathsf{CT}_1, \mathsf{CT}_2)$.

Note that the Tensor^t operation discards more components than the Tensor^2 operation from Section 4.1. The following lemma formalizes the relationship between Tensor^t and Tensor.

Lemma 5.2. *Let* $\mathsf{CT}_i = (\hat{\mathsf{ct}}_{i,0}, \cdots, \hat{\mathsf{ct}}_{i,t-1}) \in (R^2_{Q_\ell})^t$ *be a ciphertext tuple satisfying* $\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT}_i) = \mathsf{ct}_i$ *for* $i \in \{1, 2\}$. *Let* $\mathsf{sk} = (1, s) \in R^2$ *be a secret key. Then, modulo* Q_ℓ:

$$(\mathsf{ct}_1 \otimes \mathsf{ct}_2) \cdot (1, s, s^2) = q_{\mathsf{div}}^{t-1} \cdot (\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT}_1 \otimes^t \mathsf{CT}_2)) \cdot (1, s, s^2) + \sum_{i=1}^{t-1}\sum_{j=i}^{t-1} q_{\mathsf{div}}^{t-1-i} (\hat{\mathsf{ct}}_{1,j} \cdot \mathsf{sk}) \cdot (\hat{\mathsf{ct}}_{2,t-1-j} \cdot \mathsf{sk}).$$

Now, assume that $\|\mathsf{Dec}(\hat{\mathsf{ct}}_{i,0})\| \le \hat{M}$ *and* $\|\mathsf{Dec}(\hat{\mathsf{ct}}_{i,j})\| \le \check{M}$ *for all* $i \in \{1, 2\}$, $j \in \{1, \cdots, t-1\}$ *and for some* $\hat{M}, \check{M}$ *satisfying* $N(\hat{M} q_{\mathsf{div}}^{t-1} + \check{M} q_{\mathsf{div}}^{t-2} + \cdots + \check{M})^2 < Q_\ell/2$. *Then we have:*

$$\Big\| [\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT}_1 \otimes^t \mathsf{CT}_2) \cdot (1, s, s^2)]_{Q_\ell} - \frac{1}{q_{\mathsf{div}}^{t-1}} [(\mathsf{ct}_1 \otimes \mathsf{ct}_2) \cdot (1, s, s^2)]_{Q_\ell} \Big\|_\infty \le \sum_{i=1}^{t-1} \frac{(t-i)N\check{M}^2}{q_{\mathsf{div}}^i}.$$

Similarly to Lemma 4.2, Lemma 5.2 states that Tensor^t applied to CT_1 and CT_2 decrypts to approximately the same plaintext as Tensor of ct_1 and ct_2 divided by q_{div}^{t-1}, over R, when the underlying plaintexts are sufficiently small.

Definition 5.3 (Tuple relinearize). *Let* $\mathsf{CT} = (\hat{\mathsf{ct}}_0, \cdots, \hat{\mathsf{ct}}_{t-1}) \in (R^3_{Q_\ell})^t$ *be an output of* Tensor^t. *The relinearization of* CT *is defined recursively as*

$$\mathsf{Relin}^t(\mathsf{CT}) = \mathsf{DCP}^t_{q_{\mathsf{div}}}(\mathsf{Relin}(q_{\mathsf{div}}^{t-1} \cdot \hat{\mathsf{ct}}_0)) + (0, \mathsf{Relin}^{t-1}(\overline{\mathsf{CT}})),$$

where $\overline{\mathsf{CT}} = (\hat{\mathsf{ct}}_1, \cdots, \hat{\mathsf{ct}}_{t-1}) \in (R^3_{Q_\ell})^{t-1}$.

Observe that

$$\begin{aligned} &\mathsf{Relin}(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\hat{\mathsf{ct}}_0, \cdots, \hat{\mathsf{ct}}_{t-1})) \\ &= \mathsf{Relin}(q_{\mathsf{div}}^{t-1} \cdot \hat{\mathsf{ct}}_0 + \cdots + q_{\mathsf{div}} \cdot \hat{\mathsf{ct}}_{t-2} + \hat{\mathsf{ct}}_{t-1}) \\ &\approx \mathsf{Relin}(q_{\mathsf{div}}^{t-1} \cdot \hat{\mathsf{ct}}_0) + \cdots + \mathsf{Relin}(q_{\mathsf{div}} \cdot \hat{\mathsf{ct}}_{t-2}) + \mathsf{Relin}(\hat{\mathsf{ct}}_{t-1}). \end{aligned}$$

This gives an approximate identity

$$\mathsf{DCP}^t_{q_{\mathsf{div}}} \circ \mathsf{Relin} \circ \mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT}) \approx \mathsf{Relin}^t(\mathsf{CT})$$

This leads to the following result.

Lemma 5.4. *Let* $\mathsf{CT} = (\hat{\mathsf{ct}}_0, \cdots, \hat{\mathsf{ct}}_{t-1}) \in (R^3_{Q_\ell})^t$ *be an output of* Tensor^t. *Let* $\mathsf{sk} = (1, s) \in R^2$ *be a secret key with* s *of Hamming weight* h. *Then the quantity*

$$\Big[\Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{Relin}^t(\mathsf{CT}))\Big) \cdot (1, s) - \Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT})\Big) \cdot (1, s, s^2)\Big]_{Q_\ell}$$

has infinity norm $\le E_{\mathsf{Relin}} + th/2$. *Now, assume that* $\|[\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT}) \cdot (1, s, s^2)]_{Q_\ell}\|_\infty \le M$ *for some* M *satisfying* $2(M + E_{\mathsf{Relin}} + th/2) < Q_\ell/2$. *Then*

$$\Big[\Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{Relin}^t(\mathsf{CT}))\Big) \cdot (1, s)\Big]_{Q_\ell} - \Big[\Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT})\Big) \cdot (1, s, s^2)\Big]_{Q_\ell}$$

also has infinity norm $\le E_{\mathsf{Relin}} + th/2$.

Definition 5.5 (Tuple Rescale). *Let* $\mathsf{CT} = (\hat{\mathsf{ct}}_0, \cdots, \hat{\mathsf{ct}}_{t-1}) \in (R^2_{Q_\ell})^t$ *be a ciphertext tuple. Let* $q_\ell = Q_\ell / Q_{\ell-1}$. *The rescale of* CT *is defined as* $\mathsf{RS}^t_{q_\ell}(\mathsf{CT}) = (\hat{\mathsf{ct}}_{rs}, \cdots, \hat{\mathsf{ct}}_{rs,t-1}) \in (R^2_{Q_{\ell-1}})^t$ *with* $\hat{\mathsf{ct}}_{rs,0} = \mathsf{RS}_{q_\ell}(\hat{\mathsf{ct}}_0)$ *and, for* $i \in \{1, 2, \cdots, t-1\}$,

$$\begin{aligned} \hat{\mathsf{ct}}_{rs,i} = &\mathsf{RS}_{q_\ell}(q_{\mathsf{div}}^i \cdot \hat{\mathsf{ct}}_0 + q_{\mathsf{div}}^{i-1} \cdot \hat{\mathsf{ct}}_1 + \cdots + \hat{\mathsf{ct}}_i) \\ &- q_{\mathsf{div}} \cdot \mathsf{RS}_{q_\ell}(q_{\mathsf{div}}^{i-1} \cdot \hat{\mathsf{ct}}_0 + q_{\mathsf{div}}^{i-2} \cdot \hat{\mathsf{ct}}_1 + \cdots + \hat{\mathsf{ct}}_{i-1}). \end{aligned}$$

Note that the following equality holds:

$$\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{RS}^t_{q_\ell}(\mathsf{CT})) = \mathsf{RS}_{q_\ell}(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT})).$$

Lemma 5.6. *Let* $\mathsf{CT} \in (R^2_{Q_\ell})^t$ *be a ciphertext tuple. Let* $\mathsf{sk} = (1, s) \in R^2$ *be a secret key with* s *of Hamming weight* h. *Then the quantity*

$$\Big[\Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{RS}^t_{q_\ell}(\mathsf{CT}))\Big) \cdot (1, s)\Big]_{Q_{\ell-1}} - \frac{1}{q_\ell}\Big[\Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT})\Big) \cdot (1, s)\Big]_{Q_\ell}$$

has infinity norm $\le (h+1)/2$.

5.2 Multiplication for tuple representation

We can now define t-tuple multiplication.

Definition 5.7 (Tuple multiply). *We define multiplication of ciphertext tuples over* Q_ℓ *as* $\mathsf{Mult}^t := \mathsf{RS}^t_{q_\ell} \circ \mathsf{Relin}^t \circ \mathsf{Tensor}^t$, *where* $q_\ell = Q_\ell / Q_{\ell-1}$. *The result is a ciphertext pair over modulus* $Q_{\ell-1}$.

By combining Lemmas 5.2, 5.4 and 5.6, we obtain the following theorem.

Theorem 5.8. *Let* $\mathsf{CT}_i = (\hat{\mathsf{ct}}_{i,0}, \cdots, \hat{\mathsf{ct}}_{i,t-1}) \in (R^2_{Q_\ell})^t$ *be a ciphertext tuple for* $i \in \{1, 2\}$. *Let* $q_\ell = Q_\ell / Q_{\ell-1}$ *and* $\mathsf{sk} = (1, s) \in R^2$ *be a secret key with* s *of Hamming weight* h. *Assume that* $\|\mathsf{Dec}(\hat{\mathsf{ct}}_{i,0})\|_\infty \le \hat{M}$ *and* $\|\mathsf{Dec}(\hat{\mathsf{ct}}_{i,j})\|_\infty \le \check{M}$ *for all* $i \in \{1, 2\}$, $j \in \{1, \cdots, t-1\}$ *and for some* $\hat{M}, \check{M}$ *satisfying* $N(\hat{M} q_{\mathsf{div}}^{t-1} + \check{M} \cdot q_{\mathsf{div}}^{t-2} + \cdots + \check{M})^2 + E_{\mathsf{Relin}} + \frac{t}{2}h < Q_\ell/2$. *Then*

$$\Big[\Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{Mult}^t(\mathsf{CT}_1, \mathsf{CT}_2))\Big) \cdot \mathsf{sk}\Big]_{Q_{\ell-1}} - \frac{1}{q_\ell}\Big[\Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT}_1) \cdot \mathsf{sk}\Big) \cdot \Big(\mathsf{RCB}^t_{q_{\mathsf{div}}}(\mathsf{CT}_2) \cdot \mathsf{sk}\Big)\Big]_{Q_\ell}$$

has infinity norm

$$\le \Big(\sum_{i=1}^{t-1} \frac{(t-i)N\check{M}^2}{q_{\mathsf{div}}^i} + E_{\mathsf{Relin}} + \frac{t}{2}h\Big) \cdot \frac{1}{q_\ell} + \frac{h+1}{2}.$$

The correctness holds up to an error term whose main component is $(t-1)N\check{M}^2/(q_{\mathsf{div}}q_\ell)$. Since this term increases with the tuple length t, the growth of the quantity $\check{M}$ should be more carefully bounded through homomorphic computations involving sequential multiplications based on Mult^t.

Modulus consumption. We discuss about modulus consumption of Mult^t similar to Mult^2. Note that the size of $\check{M}$ is still $\approx Cq_{\mathsf{div}}^2$ where $C > 1$ is a small constant. In order to keep the main component $(t-1)N\check{M}^2/(q_{\mathsf{div}}q_\ell) \approx (t-1)NC^2 \cdot q_{\mathsf{div}}/q_\ell$, the modulus q_ℓ should be at least as large as a small multiple of q_{div}. Since we divide the size of plaintext by q_{div}^{t-1} in Tensor^t and rescale by q_ℓ in $\mathsf{RS}_{q_\ell}^t$, the scale is divided by $q_{\mathsf{div}}^{t-1}q_\ell$ after every multiplication. Hence we have a relation $\Delta \simeq q_{\mathsf{div}}^{t-1}q_\ell$. Since we chose q_ℓ to be slightly larger than q_{div}, this relation allows us to choose Δ to be as large as $\approx q_\ell^t$. The main advantage of Mult^t over classical CKKS homomorphic multiplication is that for the same modulus consumption we obtain a t-multiple precision computation on plaintexts.

Efficiency. In terms of run-time, executing Mult^t requires $t(t+1)/2$ calls to Tensor, t calls to Relin and t calls to RS. Note that only Relin and RS involve NTTs, so that the contribution of Tensor to the overall cost is negligible when t is $o(\log N)$. Therefore, the multiplication time is proportional to t. Both Relin and RS have a costs that are quasi-linear in the bit-size of the working modulus. However, Mult^t requires moduli that are t-times smaller, the costs of Mult and Mult^t are also similar for performing a sequence of computations. In fact, as Mult^t requires smaller moduli, one can also use smaller-degree rings for the same security, which gives it an advantage in terms of latency.

In terms of swk bit-size, Mult^t outperforms Mult by a factor $\approx t$ due to the smaller moduli that it involves. As for run-time, this impact further increases if one takes into account that smaller moduli enable the use of smaller-degree rings.

5.3 Bounding the low parts

In Theorem 5.8, we observed that the infinity norm of the error of pair multiplications is bounded by a function of $\check{M}$, an upper bound of $\|\mathsf{Dec}(\hat{\mathsf{ct}}_{i,j})\|_\infty$ for $i \in \{1,2\}$ and $j \in \{1,\cdots,t-1\}$. As the decryption of all low parts $\|\mathsf{Dec}(\hat{\mathsf{ct}}_j)\|_\infty$ for all $j \in \{1,\cdots,t-1\}$ increases as we proceed with a sequence of multiplications, the multiplication error also grows. Since Theorem 3.2 gives the initial upper bound of all low parts right after the first decomposition, it suffices to analyze the growth of all low parts through a single multiplication (the analysis for addition and subtraction is elementary, and key switching is handled identically to the relinearization step of multiplication). Similarly to Section 4.3, we use the canonical embedding to get tighter bounds.

Theorem 5.9 (Growth of low parts). *Let* $\mathsf{CT}_i = (\hat{\mathsf{ct}}_{1,0},\cdots,\hat{\mathsf{ct}}_{i,t-1}) \in (R_{Q_\ell}^2)^t$ *be a ciphertext tuple for* $i \in \{1,2\}$. *Let* $q_\ell = Q_\ell/Q_{\ell-1}$ *and* $\mathsf{sk} = (1,s) \in R^2$ *be a secret key with* s *of Hamming weight* h. *Suppose that* $\|\mathsf{can}\circ\mathsf{Dec}(\hat{\mathsf{ct}}_{i,0})\|_\infty \le \hat{M}$ *and* $\|\mathsf{can}\circ\mathsf{Dec}(\hat{\mathsf{ct}}_{i,j})\|_\infty < \check{M}$ *for* $i \in \{1,2\}$, $j \in \{1,\cdots,t-1\}$ *and for some* $\hat{M},\check{M}$. *Let* $\mathsf{CT}_{\mathsf{Mult}} = \mathsf{Mult}^t(\mathsf{CT}_1,\mathsf{CT}_2) = (\hat{\mathsf{ct}}_{\mathsf{Mult},0},\cdots,\hat{\mathsf{ct}}_{\mathsf{Mult},t-1})$. *Then the following holds:*

$$\|\mathsf{can}\circ\mathsf{Dec}(\hat{\mathsf{ct}}_{\mathsf{Mult},j})\|_\infty \le \frac{2\hat{M}\check{M}+(j-1)\check{M}^2}{q_\ell} + N\Big(\frac{E_{\mathsf{Relin}}}{q_\ell} + j\cdot(h+3)(q_{\mathsf{div}}+1)\Big),$$

where $j \in \{1,\cdots,t-1\}$.

The main term of the upper bound is $(2\hat{M}\check{M}+(t-2)\check{M}^2)/q_\ell$, which is proportional to $\check{M}$. In the classical CKKS scheme, we usually assume that $\|\mathsf{Dcd}\circ\mathsf{Dec}(\mathsf{ct})\|_\infty \le 1$. Since $\mathsf{Dcd} = \Delta^{-1}\cdot\mathsf{can}$, this can be reinterpreted as $\|\mathsf{can}\circ\mathsf{Dec}(\mathsf{ct})\|_\infty \le \Delta$. Hence, we can assume that $\hat{M} = \Delta/q_{\mathsf{div}}^{t-1} \approx q_\ell$. Also, we could choose q_ℓ to be slightly larger than q_{div}, we can assume that $\hat{M}$ is larger than $\check{M}$. In this case, the main term is as large as $t\Delta\check{M}/(q_{\mathsf{div}}^{t-1}q_\ell) \approx t\check{M}$. Therefore, the upper bound of all low parts grows approximately by $\log t$ bits after each multiplication.

6 EXPERIMENTS

We conducted experiments based on a proof-of-concept implementation of Mult^2 (i.e., $t = 2$). Our code is developed upon the C++ HEaaN library.[5] The experiments are conducted on an Intel Xeon Gold 6242 at 2.8 GHz with 503GiB of RAM running Linux. All security estimates derive from [1, 15].

Our implementation relies on RNS arithmetic. For the rescaling function RS (which is used in both Quo and Relin^2), we proceed exactly as in RNS-based rescaling [11, 18], with an extra RNS prime equal to q_{div} (if q_{div} is larger than 2^{64}, one can set it as the product of several RNS primes, but this was the case in none of our experiments). For high-precision computations, such as with a scaling factor Δ of 100 bits, the list of RNS primes in our approach differs from what would be done with RNS-based CKKS. Typically, one would set the multiplication-ladder moduli q_i as products of two RNS primes: $q_i = q_{i1}\cdot q_{i2} \approx \Delta$, with q_{i1} and q_{i2} primes that both fit in a 64-bit word. In our case, we only need $q_{\mathsf{div}}\cdot q_i \approx \Delta$ and can set q_{div} and all q_i's to be primes that fit in a 64-bit word. This implies that each modulus q_i corresponds to a single RNS prime.

We recall some notations which are extensively used in this section: N denotes the ring degree, h the secret key Hamming weight, Δ the scaling factor and Q_LP the largest switching key modulus.

6.1 Error growth

Table 1 describes the parameters we used in our first experiment. We considered two parameter sets with $t = 1$ (designed for the CKKS Mult algorithm) and $t = 2$ (designed for the Mult^2 algorithm), in order to compare the error growths of both approaches. The parameter sets are designed so that a similar precision, a similar decryption capacity and a similar security (of ≈ 128 bits) are reached. Further, for both of them, we use the same number of multiplication levels (equal to 8). As expected, the overall largest modulus Q_LP is smaller for $t = 2$ than for $t = 1$, although not by a factor 2, as asymptotically lower-order terms contribute significantly for the relatively small bit-sizes we consider.

[5] We are using the CryptoLab HEaaN library, available at https://www.heaan.it/.

Table 1: Parameters for the error growth observation. Here $\log_2 q$ denotes the bit-sizes of prime factors in the modulus chains, and $\log_2 P$ denotes the size of the auxiliary switching key primes. Base **refers to the base prime Q_0,** Mult **refers to the multiplication primes $q_\ell = Q_\ell / Q_{\ell-1}$ (the table entry also provides the number L of such primes), and** Div **refers to the modulus q_{div} used in Mult^2. The secret key has Hamming weight $h = 21,845$.**

Multiplication algorithm	N	$\log_2(Q_L P)$	$\log_2 \Delta$	$\log_2 q$			$\log_2 P$
				Base	Mult	Div	
Mult ($t = 1$)	2^{15}	610	61	61	61×8	–	61
Mult^2 ($t = 2$, new)		449		61	38×8	23	61

We performed 8 repeated squarings on a single ciphertext, and measured how the error grows as a function of the number of multiplications, as visualized in Figure 1.

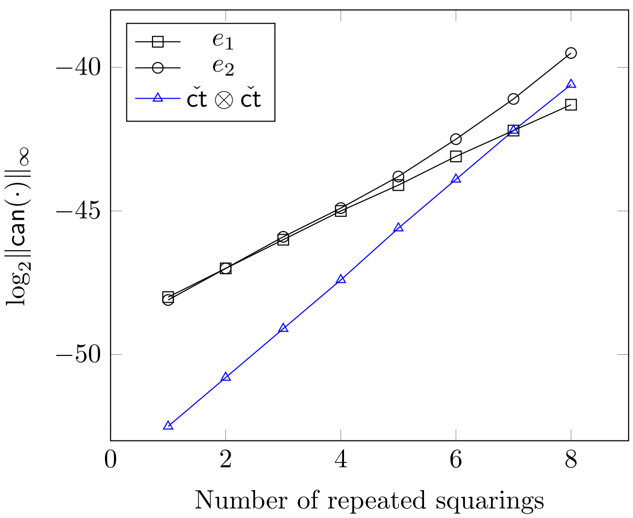

Figure 1: Error growth as a function of the number of repeated squarings. The data points for e_1 and e_2 are the average magnitudes of the infinite norms of errors for $t = 1$ and $t = 2$, respectively. The data points for $\check{\mathsf{ct}} \otimes \check{\mathsf{ct}}$ correspond to the average magnitudes of the squared lower parts after rescaling them by q_{div}. The norms are with respect to the canonical embedding. The averages are taken over 1,000 executions.

The multiplication error e_1 in the usual CKKS setting increases by at most 1 bit after every squaring. On the other hand, when using Mult^2, the square of the lower part $\check{\mathsf{ct}} \otimes \check{\mathsf{ct}}$ increases by approximately 1.7 bits per multiplication. This corresponds to the term that was discarded in the tensor step Tensor^2, and contributes to e_2. Consistently with Theorem 4.8, the magnitude of e_2 is essentially the sum of the magnitudes of e_1 and $\check{\mathsf{ct}} \otimes \check{\mathsf{ct}}$. Therefore, at the beginning, the multiplication e_2 follows the growth of e_1. As $\check{\mathsf{ct}} \otimes \check{\mathsf{ct}}$ grows, the quantity e_2 starts to increase faster than e_1. In our experiment, this becomes significant at the 6th repeated squaring.

6.2 Increased homomorphic capacity

We now aim at maximizing multiplicative depth, for a given ring degree (set here to $N = 2^{15}$). We designed two parameter sets, for $t = 1$ and $t = 2$. Both parameter sets used the largest modulus possible $Q_L P$ while retaining 128 bits of security. The detailed parameter sets are provided in Table 2. For a fair comparison, we decreased the moduli for $t = 1$ (from 61 in Table 1 to 57 in Table 2), so that both parameter sets in Table 2 lead to similar numerical errors. Indeed, as seen in the previous subsection, Mult^2 degrades the numerical accuracy slightly faster. This adjustment brings one more level to the $t = 1$ parameter set. Overall, the second parameter set provides a significantly larger multiplicative depth (18) than the first one (13).

Table 2: Parameters maximizing the multiplicative depth while retaining 128 bits of security. The secret key has Hamming weight $h = 21,845$.

Mult. algorithm	N	$\log_2(Q_L P)$	$\log_2 \Delta$	$\log_2 q$			$\log_2 P$
				Base	Mult	Div	
Mult ($t = 1$)	2^{15}	855	57	57	57×13	–	57
Mult^2 ($t = 2$, new)		875	61	61	38×18	23×3	61

As the moduli in the $t = 2$ parameter sets of Tables 1 and 2 are the same, we expect the error to grow as depicted in Figure 1. In particular, after 6 multiplication layers, one expects e_2 to start growing at an increased pace. To thwart this phenomenon, we run recombine and decompose ($\mathsf{DCP} \circ \mathsf{RCB}$), in order to decrease $\check{\mathsf{ct}} \otimes \check{\mathsf{ct}}$ and maintain the precision. Concretely, we perform this error refreshing between multiplication levels 6 and 7 and between multiplication levels 12 and 13 (out of 18 levels). This explains the use of 3 Div primes in the $t = 2$ parameter set: 1 Div prime for the initial DCP and 2 Div primes for the error refreshings.

To observe the error growth of these two parameter sets, we performed 18 repeated squarings on a single ciphertext and measured the average infinity norms of errors over 1,000 executions. As the parameter with $t = 1$ has multiplicative depth smaller than 18, we constructed a parameter set with extended multiplicative depth equal to 18 only to measure the precision, with $\log_2(Q_L P) = 855 + 5 \cdot 57 = 1,140$ (note that for such a large $Q_L P$, the parameter set would not reach 128 bits of security). Parameters with $t = 1$ and $t = 2$ showed -31.3 bits and -31.0 bits of precision, respectively.

We now study the gain of Mult^2 over Mult, taking error refreshing (i.e., recombine and decompose) into account.

Let k be the number of Mult^2 sequential multiplications that can be performed between two consecutive error refreshings. Such a block of operations consists of 1 decompose and k multiplications, consuming an amount $q_{\mathsf{div}} \cdot (\Delta / q_{\mathsf{div}})^k$ of modulus. The maximum multiplication depth starting from modulus Q can be computed as

$$\frac{\log_2(Q)}{\log_2(\Delta) - (1 - 1/k) \cdot \log_2(q_{\text{div}})}.$$

For comparison, recall that the multiplicative depth of original Mult is $\log_2(Q)/\log_2(\Delta)$. The depth gain of Mult^2 over Mult is just the difference between these two multiplicative depths. At a high level, when k is sufficiently large and q_{div} is set to $\approx \sqrt{\Delta}$, we expect a factor 2 improvement.

We now explain a strategy to choose k. According to Theorem 4.9, the low part grows by approximately 1 bit after each multiplication. When we get a ciphertext pair from a ciphertext by decomposing it, the low part has infinity norm $\leq q_{\text{div}} \cdot (h+1)/2$. In order to maintain the error coming from the low part smaller than the desired bound E, it suffices that

$$\frac{N \cdot ((h+1) \cdot q_{\text{div}} \cdot 2^{k-2})^2}{\Delta} < E,$$

thanks to Theorem 4.8. One can choose k to be the largest integer satisfying

$$k \leq \frac{\log_2(E) + \log_2(\Delta) - \log_2(N)}{2} - \log_2(q_{\text{div}}) - \log_2(h+1) + 2.$$

Note that $\log_2(\Delta)$ and $\log_2(q_{\text{div}})$ are the most significant terms here. We suggest to choose q_{div} slightly smaller than $\sqrt{\Delta}$ so that we can have sufficiently large k. Experiments show that this analysis is pessimistic and allow for better pairs (k, q_{div}).

6.3 Increased precision

We now consider a setting where we want to perform a homomorphic computation on plaintexts that have large bit-sizes. This is a situation that can come up for specific applications involving large numbers or high precision, as well as for specific applications requiring $\text{IND-CPA}^{\text{D}}$ security [34, 35]. In this third experimental setup, we consider multiplicative depth equal to 8 and plaintexts of 100 bits, i.e., about twice larger than modulus bit-sizes typically considered with CKKS (for efficiency purposes, it is convenient to set the modulus so that it fits within a 64-bit machine word).

The classical approach would consist in batching Base and Mult moduli by pairs, so that each pair product approximately matches with $\Delta = 2^{100}$. With 8 levels, this leads to a large maximal modulus $Q_L P$ of 1,000 bits. As this is more than the maximal modulus allowed for ring degree $N = 2^{15}$ with 128 bits of security, this leads to choosing degree $N = 2^{16}$. This gives the $t = 1$ parameter set of Table 3. In the table, the '(50×2)' notation means that two 50-bit primes are being paired to have a product that matches $\Delta = 2^{100}$.

Let us now explain how to use Double-CKKS in this context, i.e., with $t = 2$. We can decompose the scaling factor Δ as $q \simeq 2^{60}$ and $q_{\text{div}} \simeq 2^{40}$. Even though the multiplicative depth is maintained, this allows to greatly reduce the maximum modulus $Q_L P$. In turn, this allows to halve the ring degree, from $N = 2^{16}$ to $N = 2^{15}$ while retaining 128 bits of security. This corresponds to the $t = 2$ parameter set of Table 3. The margin between $q_{\text{div}} \simeq 2^{40}$ and $q \simeq 2^{60}$ is quite large, so that we can maintain precision even without the recombine and decompose strategy mentioned in Section 6.2. To check precision, we performed 8 repeated squarings on a single ciphertext and measured the average infinity norms of errors over 1,000 executions. We obtained -81.2 bit and -81.8 bit precision for the $t = 1$ and $t = 2$ parameters, respectively.

We now focus on efficiency. The multiplication latency decreased from 270ms to 179ms, the ciphertext size (for the largest modulus) decreased from 14.8MB to 5.08MB, and the switching key size (for the largest modulus) decreased from 74MB to 30.6MB: i.e., 1.5 times faster multiplication, 2.9 times smaller ciphertexts, and 2.4 times smaller switching keys.[6] This stems from the lower moduli and lower ring degree N. Note that the number of 64-bit unit NTTs is 290 and 350, respectively, but the former has an NTT dimension that is twice than the latter.

Table 3: Comparison of two approaches for computations with large precision. Here *dnum* denotes the key switching gadget rank [3, 20, 21], T_{mult} denotes the multiplication time (using a single thread), *#NTT* denotes the number of 64-bit unit NTTs in dimension N, and ctxt size **and** key size **respectively denote the size of a ciphertext and a single switching key at the maximum level. The secret key has Hamming weight $h = 128$.**

Mult. algorithm	N	*dnum*	T_{mult}	*#NTT*	ctxt size	key size
Mult ($t = 1$)	2^{16}	9	270ms	290	14.8MB	73.7MB
Mult^2 ($t = 2$, new)	2^{15}	11	**179ms**	**350**	**5.08MB**	**30.6MB**

$\log_2(Q_L P)$	$\log_2 \Delta$	$\log_2 q$			$\log_2 P$
		Base	Mult	Div	
1,000	100	(50×2)	$(50 \times 2) \times 8$	-	(50×2)
680		(50×2)	60×8	40	60

7 CONCLUSION

Our new homomorphic multiplication, Mult^t, decomposes a high precision homomorphic multiplication into lower precision ones, reducing the modulus consumption by roughly a factor t. This allows to increase homomorphic capacity and to increase precision without increasing parameters.

In this work, we did not implement tuple-CKKS for $t > 2$ because we focused on examples with moderate precision. One could be interested in $t > 2$ for higher precision scenarios, and we leave the experimental aspects of $t > 2$ open for future work. Note that obtaining an efficient implementation for $t > 2$ is not straightforward, since arithmetic modulo q_{div}^{t-1} is not directly compatible with RNS-CKKS. We expect that one could modify the RNS representation to handle q_{div}^{t-1} separately while keeping RNS for the other moduli. Such an implementation would be useful to assess the impact of t on the efficiency and accuracy.

Although we discussed the possibility of bootstrapping using Mult^t for $t > 1$, we did not provide an implementation of it. In order to construct an efficient pair/tuple bootstrapping, one needs to carefully manage error growth especially in homomorphic linear

[6] For ciphertext size, we do not consider the possibility of storing the right side as the seed of a hash function, as this is possible only at the largest modulus. For switching key size, we consider this representation.

transformations, and also deal with the use of different scaling factors during bootstrapping. We also leave this open for future work.

REFERENCES

[1] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2019. Homomorphic Encryption Standard. Cryptology ePrint Archive, Paper 2019/939. https://eprint.iacr.org/2019/939

[2] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. 2022. META-BTS: Bootstrapping Precision Beyond the Limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 223–234. https://doi.org/10.1145/3548606.3560696

[3] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. 2017. A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes. In *Selected Areas in Cryptography – SAC 2016*, Roberto Avanzi and Howard Heys (Eds.). Springer International Publishing, Cham, 423–442.

[4] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. 2018. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. In *Advances in Cryptology – CRYPTO 2018*. Springer International Publishing, Cham, 565–596.

[5] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 868–886.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) Fully Homomorphic Encryption without Bootstrapping *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 309–325. https://doi.org/10.1145/2090236.2090262

[7] Nathan Manohar Charanjit S. Jutla. 2022. Sine Series Approximation of the Mod Function for Bootstrapping of Approximate HE. In *Advances in Cryptology – EUROCRYPT 2022*.

[8] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT 2019*, Yuval Ishai and Vincent Rijmen (Eds.). Springer International Publishing, Cham, 34–54.

[9] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2019. Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 395–412. https://doi.org/10.1145/3319535.3363207

[10] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology – EUROCRYPT 2018*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer International Publishing, Cham, 360–384.

[11] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography – SAC 2018 (Lecture Notes in Computer Science, Vol. 11349)*. Springer, 347–368.

[12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 409–437.

[13] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *Advances in Cryptology – ASIACRYPT 2016*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–33.

[14] Michael Clear and Ciaran McGoldrick. 2015. Multi-identity and Multi-key Leveled FHE from Learning with Errors. In *Advances in Cryptology - CRYPTO*, Rosario Gennaro and Matthew Robshaw (Eds.). Springer, 630–656. https://doi.org/10.1007/978-3-662-48000-7_31

[15] Benjamin R. Curtis and Rachel Player. 2019. On the Feasibility and Impact of Standardising Sparse-Secret LWE Parameter Sets for Homomorphic Encryption. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (London, United Kingdom) *(WAHC'19)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3338469.3358940

[16] T. J. Dekker. 1971. A floating-point technique for extending the available precision. *Numer. Math.* 18, 3 (1971), 224–242. https://doi.org/10.1007/BF01397083

[17] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Ph. D. Dissertation. Stanford University, USA.

[18] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Homomorphic Evaluation of the AES Circuit. In *Advances in Cryptology – CRYPTO 2012 (Lecture Notes in Computer Science, Vol. 7417)*. Springer, 850–867. https://doi.org/10.1007/978-3-642-32009-5_49

[19] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology – CRYPTO 2013*, Ran Canetti and Juan A. Garay (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 75–92.

[20] Shai Halevi, Yuriy Polyakov, and Victor Shoup. 2019. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. In *Topics in Cryptology – CT-RSA 2019*, Mitsuru Matsui (Ed.). Springer International Publishing, Cham, 83–105.

[21] Kyoohyung Han and Dohyeong Ki. 2020. Better Bootstrapping for Approximate Homomorphic Encryption. In *Topics in Cryptology – CT-RSA 2020*, Stanislaw Jarecki (Ed.). Springer International Publishing, Cham, 364–390.

[22] Seungwan Hong, Jai Hyun Park, Wonhee Cho, Hyeongmin Choe, and Jung Hee Cheon. 2022. Secure tumor classification by shallow neural network using homomorphic encryption. *BMC Genomics* 23, 1 (2022). https://doi.org/10.1186/s12864-022-08469-w

[23] Mioara Joldeş, Olivier Marty, Jean-Michel Muller, and Valentina Popescu. 2016. Arithmetic Algorithms for Extended Precision Using Floating-Point Expansions. *IEEE Trans. Comput.* 65, 4 (2016), 1197–1210. https://doi.org/10.1109/TC.2015.2441714

[24] Charanjit S. Jutla and Nathan Manohar. 2020. Modular Lagrange Interpolation of the Mod Function for Bootstrapping of Approximate HE. Cryptology ePrint Archive, Report 2020/1355. https://ia.cr/2020/1355.

[25] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics* 11, 4 (2018). https://doi.org/10.1186/s12920-018-0401-7

[26] Duhyeong Kim, Yongha Son, Dongwoo Kim, Andrey Kim, Seungwan Hong, and Jung Hee Cheon. 2020. Privacy-preserving approximate gwas computation based on homomorphic encryption. *BMC Medical Genomics* 13, 7 (2020). https://doi.org/10.1186/s12920-020-0722-1

[27] Miran Kim, Arif Ozgun Harmanci, Jean-Philippe Bossuat, Sergiu Carpov, Jung Hee Cheon, Ilaria Chillotti, Wonhee Cho, David Froelicher, Nicolas Gama, Mariya Georgieva, Seungwan Hong, Jean-Pierre Hubaux, Duhyeong Kim, Kristin Lauter, Yiping Ma, Lucila Ohno-Machado, Heidi Sofia, Yongha Son, Yongsoo Song, Juan Troncoso-Pastoriza, and Xiaoqian Jiang. 2021. Ultrafast homomorphic encryption models enable secure outsourcing of genotype imputation. *Cell Systems* 12, 11 (2021), 1108–1120.e4. https://doi.org/10.1016/j.cels.2021.07.010

[28] Miran Kim, Yongsoo Song, Baiyu Li, and Daniele Micciancio. 2020. Semi-parallel logistic regression for GWAS on encrypted data. *BMC Medical Genomics* 13, 7 (2020).

[29] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 12403–12422. https://proceedings.mlr.press/v162/lee22e.html

[30] Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* 10 (2022), 30039–30054. https://doi.org/10.1109/ACCESS.2022.3159694

[31] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 2021. High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function. In *Advances in Cryptology – EUROCRYPT 2021*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer International Publishing, Cham, 618–647.

[32] Joon-Woo Lee, Yongwoo Lee, Young-Sik Kim, Youngjune Kim, Jong-Seon No, and HyungChul Kang. 2022. High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization. In *Advances in Cryptology – EUROCRYPT 2022*.

[33] Yongwoo Lee, Joon-Woo Lee, Young-Sik Kim, and Jong-Seon No. 2020. Near-Optimal Polynomial for Modulus Reduction Using L2-Norm for Approximate Homomorphic Encryption. *IEEE Access* PP (08 2020), 1–1. https://doi.org/10.1109/ACCESS.2020.3014369

[34] Baiyu Li and Daniele Micciancio. 2021. On the Security of Homomorphic Encryption on Approximate Numbers. In *Advances in Cryptology – EUROCRYPT 2021*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer International Publishing, Cham, 648–677.

[35] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. 2022. Securing Approximate Homomorphic Encryption Using Differential Privacy. In *Advances in Cryptology - CRYPTO 2022*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, 560–589. https://doi.org/10.1007/978-3-031-15802-5_20

[36] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 2017. Multikey Fully Homomorphic Encryption and Applications. *SIAM J. Comput.* 46, 6 (2017), 1827–1892. https://doi.org/10.1137/14100124X

[37] Qian Lou and Lei Jiang. 2021. HEMET: A Homomorphic-Encryption-Friendly Privacy-Preserving Mobile Neural Network Architecture. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 7102–7110. http://proceedings.mlr.press/v139/lou21a.html

[38] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. A Toolkit for Ring-LWE Cryptography. In *Advances in Cryptology – EUROCRYPT 2013*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 35–54.

[39] Anisha Mukherjee, Aikata Aikata, Ahmet Can Mert, Yongwoo Lee, Sunmin Kwon, Maxim Deryabin, and Sujoy Sinha Roy. 2023. ModHE: Modular Homomorphic Encryption Using Module Lattices: Potentials and Limitations. Cryptology ePrint Archive, Paper 2023/895. https://eprint.iacr.org/2023/895 https://eprint.iacr.org/2023/895.

[40] SEAL 2020. Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..