Plaintext-Ciphertext Matrix Multiplication and FHE Bootstrapping: Fast and Fused

Youngjin Bae[1][0000−0001−6870−4504], Jung Hee Cheon[1,2][0000−0002−7085−2220], Guillaume Hanrot[3][0000−0001−9319−0365], Jai Hyun Park[2][0000−0002−5401−8949], and Damien Stehlé[3][0000−0003−3435−2453]

[1] CryptoLab Inc., Seoul, Republic of Korea
[2] Seoul National University, Seoul, Republic of Korea
[3] CryptoLab Inc., Lyon, France

Abstract. Homomorphically multiplying a plaintext matrix with a ciphertext matrix (PC-MM) is a central task for the private evaluation of transformers, commonly used for large language models. We provide several RLWE-based algorithms for PC-MM that consist of multiplications of plaintext matrices (PP-MM) and comparatively cheap pre-processing and post-processing steps: for small and large dimensions compared to the RLWE ring degree, and with and without precomputation. For the algorithms with precomputation, we show how to perform a PC-MM with a single floating-point PP-MM of the same dimensions. This is particularly meaningful for practical purposes as a floating-point PP-MM can be implemented using high-performance BLAS libraries.
The algorithms rely on the multi-secret variant of RLWE, which allows to represent multiple ciphertexts more compactly. We give algorithms to convert from usual shared-secret RLWE ciphertexts to multi-secret ciphertexts and back. Further, we show that this format is compatible with homomorphic addition, plaintext-ciphertext multiplication, and key-switching. This in turn allows us to accelerate the slots-to-coeffs and coeffs-to-slots steps of CKKS bootstrapping when several ciphertexts are bootstrapped at once. Combining batch-bootstrapping with efficient PC-MM results in MaMBo(Matrix Multiplication Bootstrapping), a bootstrapping algorithm that can perform a PC-MM for a limited overhead.

1 Introduction

Plaintext-ciphertext homomorphic matrix multiplication (PC-MM) takes as inputs a matrix $\mathbf{U} = (u_{i,j})_{0 \le i < d_1, 0 \le j < d_2}$ (in clear) and ciphertext(s) corresponding to a matrix $\mathbf{M} = (m_{i,j})_{0 \le i < d_2, 0 \le j < d_3}$ and returns ciphertext(s) corresponding to the matrix $\mathbf{U} \cdot \mathbf{M}$. In this work, we will focus on matrices over $\mathbb{R}$ with approximate computations, even though most of our techniques carry over to other rings, such as $\mathbb{Z}_p$ for an arbitrary integer p. PC-MM arises in the context of privacy-preserving machine learning, notably in the inference phase of transformers for large language models (LLM), such as GPT [32], BERT [13] and

LLaMA [34]. PC-MM has been applied for the private evaluation of such large language models along with secure multiparty computation [22, 29, 14] or fully homomorphic encryption [35]. In [18], PC-MM is used for federated principal component analysis between different data providers, and, in [26], applications to smart contracts, health and finance are described.

For many of the applications above, PC-MM should handle variable matrix dimensions. In LLM inference, large-dimensional matrix multiplications appear at diverse steps. Let us temporarily delve into how LLMs work, to explain the use of large matrix dimensions. LLMs operate on user input, segmented into tokens, i.e., elementary language units. The processing ability of the LLM is directly related to its context size, i.e., the maximum number of tokens ntok that can be processed at a time. The choice of ntok is directly reflected in the computational cost via the dimensions of the matrix multiplications to be performed. Following the transformer model, LLMs usually have two internal dimensions: the main dimension dim and another dimension dff used in the so-called Feed-Forward phase. Typically, dff is 2 to 4 times larger than dim. Overall, LLMs perform three different types of matrix multiplications:

- in the so-called attention phase, weight matrices $\mathbf{U} \in \mathbb{R}^{\mathsf{dim}\times\mathsf{dim}}$ is multiplied by several user data matrices $\mathbf{M} \in \mathbb{R}^{\mathsf{dim}\times\mathsf{ntok}}$;
- in the feed-forward phase, larger weight matrices $\mathbf{U} \in \mathbb{R}^{\mathsf{dff}\times\mathsf{dim}}$ are multiplied by processed user data $\mathbf{M} \in \mathbb{R}^{\mathsf{dim}\times\mathsf{ntok}}$ before the activation step;
- later in the feed-forward phase, the processed data is reduced back to dimension dim, by multiplying another large weight matrix $\mathbb{R}^{\mathsf{dim}\times\mathsf{dff}}$ with the current user data matrix $\in \mathbb{R}^{\mathsf{dff}\times\mathsf{ntok}}$.

For privacy-preserving LLM inference, the weight matrices above are in clear, whereas the user data matrices are encrypted. In mainstream LLMs, the ntok dimension ranges from 128 to 16 384 (in GPT-3.5) or more, and the dim dimension ranges from 4 096 (LLaMA-7B) to 18 432 (PaLM 540B). These dimensions are much larger than those considered by most works considering PC-MM.

State of the art PC-MM algorithms rely on fully homomorphic encryption schemes (FHE) based on the ring learning with errors problem [33, 27] (RLWE), such as BGV [6], BFV [5, 16] and CKKS [10].[4] Most have asymptotic run-times that are linear in the product of the dimensions (or less for large dimensions, using recursive blocking techniques). However, their concrete performance is hindered by their use of possibly large RLWE moduli and, more importantly, the fact that they move data across RLWE coefficients or slots (i.e., coefficients of a Fourier transform), within RLWE ciphertexts. A notable example is the algorithm from [24], designed for ciphertext-ciphertext matrix multiplication but which can be adapted to PC-MM. It is used for example in [18], which reports a multiplication of a 256×256 plaintext matrix with a 256×8 ciphertext matrix in 3.8s on an Intel Xeon 2.5GHz with 24 threads on 12 cores. The algorithm

[4] In this paper, we mostly focus on CKKS scheme. However, most of our techniques can be adapted to BGV [6] and BFV [5, 16].

from [26] does not move data within RLWE ciphertexts, but requires matrix dimensions above the RLWE ring degree. In practice, to enable computations with a reasonable precision and 128-bit security, the degree should be at least 2^{12} for a stand-alone PC-MM, and 2^{13} if we want to enable key-switching for compatibility with FHE. This algorithm reduces one PC-MM to two PP-MM's modulo a moderately-sized integer. Even though no server-side experiment is reported in [26], this hints to the fact that PC-MM implementations can be based on fast linear algebra software and hence be very efficient.

Contributions. We describe several PC-MM algorithms that reduce to one or two PP-MM's, for dimensions below and above the RLWE ring degree, in a black-box manner and with limited overhead. We experimentally observe that the concrete performance can then directly benefit from highly-optimized linear algebra libraries such as OpenBLAS [28]. We also show how to integrate our PC-MM algorithms into FHE bootstrapping, which can itself be accelerated by techniques that we introduce.

Our algorithms extend the one from [26], providing greater dimension flexibility and greater efficiency. We give two algorithms, for square matrices dimensions d smaller and larger than the RLWE ring degree N (note that [26] works only for $d \geq N$). In both cases, they consist in reducing PC-MM to two PP-MM's, where the PP-MM dimensions are $d \times d \times N$ and $d \times d \times d$. We also describe two algorithms (for small and large d) which only use a single $d \times d \times d$ modular PP-MM, if one allows for precomputation. Finally, we show that the latter modular PP-MM can be replaced by floating-point PP-MM, hence obtaining reductions from PC-MM to a single floating-point PP-MM with the same dimension, with precomputation. These results are summarized in Table 1.

d	Ctxt format	Reduces to	Reference
$\leq N$	MLWE	Mod-PP-MM$_{d,d,d}$ + Mod-PP-MM$_{d,d,N}$	Alg. 2, p. 21
$\leq N$	sh.-a MLWE	Precomp. + Mod-PP-MM$_{d,d,d}$	Alg. 4, p. 38
$\leq N$	sh.-a MLWE	Precomp. + FP-PP-MM$_{d,d,d}$	Alg. 4, p. 38 + Sec. 5
$\geq N$	RLWE	2× Mod-PP-MM$_{d,d,d}$	[26]
$\geq N$	sh.-a RLWE	Mod-PP-MM$_{d,d,d}$ + Mod-PP-MM$_{d,d,N}$	Alg. 1, p. 18
$\geq N$	sh.-a RLWE	Precomp. + Mod-PP-MM$_{d,d,d}$	Alg. 3, p. 37
$\geq N$	sh.-a RLWE	Precomp. + FP-PP-MM$_{d,d,d}$	Alg. 3, p. 37 + Sec. 5

Table 1. List of PC-MM algorithms for square matrices of dimension $d \geq N^{1/2}$. sh.-a refers to the shared-a format (see Section 3). Mod-PP-MM$_{d,d,d'}$ (resp. FP-PP-MM$_{d,d,d'}$) stands for PP-MM of dimensions $(d \times d) \times (d \times d')$ over $\mathbb{Z}_Q$ (resp. over $\mathbb{R}$ using floating-point arithmetic).

Going deeper, the main new technical tool is the use of compact format for multiple RLWE ciphertexts. Recall that a RLWE ciphertext consists of a pair (a, b) of polynomials such that $a \cdot \mathsf{sk} + b \approx m$, where sk is the secret key and m is the underlying message. We say that several RLWE ciphertexts are in shared-

a format if their first components "a" are identical. This format has been used many times for *security proofs* in lattice-based cryptography, in the context of the lossy-mode proof technique (see, among many others, [31, 20, 7]). We use this format for *computation* purposes. We provide conversion algorithms from shared-secret RLWE ciphertexts to shared-a ciphertexts, and vice versa. Additionally, we show that many of the CKKS operations are compatible with this shared-a format, including addition, multiplication between a plaintext and a ciphertext, and key-switching (which implies rotations and conjugation). Importantly, these can be leveraged to batch-bootstrap several ciphertexts for a lower cost than running several bootstrappings independently. This is particularly relevant in the context of PC-MM, as matrices that are not very small require several RLWE ciphertexts.

Finally, we explain how to jointly perform PC-MM and FHE bootstrapping, obtaining MaMBo (Matrix Multiplication Boostrapping), which may be viewed as a form of programmable bootstrapping [12] that can perform a PC-MM at almost no cost.

We implemented several of our algorithms using the HEaaN library [23]. We provide experimental data that focuses on: (1) the cost of PC-MM with RLWE-based schemes and no precomputation and (2) the cost of batch bootstrapping. We then estimate the overall cost of MaMBo. Compared to the state of art CKKS bootstrapping procedure without PC-MM, MaMBo (including a PC-MM) is more than 40% faster when used to bootstrap ciphertexts corresponding to matrices of dimensions 2^9 and more.

1.1 Technical overview

For the sake of simplicity, in this overview, we assume that matrices are square. Let d denote their dimension. Let $q \geq 2$, N be a power of 2, $\mathcal{R}_N = \mathbb{Z}[X]/(X^N+1)$ and $\mathcal{R}_{q,N} = \mathbb{Z}_q[X]/(X^N+1)$. When there is no ambiguity regarding the value of N, we shall simply write $\mathcal{R}$ and $\mathcal{R}_q$.

The LZ algorithm. Our starting point is the algorithm from [26]. Assume first that $d = N$. The matrix $\mathbf{M}$ is provided by d RLWE ciphertexts (a_i, b_i), that encrypt the rows. We have, over $\mathcal{R}_q$:

$$\forall i : a_i \cdot \mathsf{sk} + b_i \;\approx\; \sum_j m_{i,j} X^j \;\;,$$

where $\mathsf{sk} \in \mathcal{R}$ is the secret key. We can rewrite the above in matrix form, as follows:

$$\mathbf{A} \cdot \mathtt{Toep}(\mathsf{sk}) + \mathbf{B} \;\approx\; \mathbf{M} \mod q \;\;, \tag{1}$$

where the i-th row of $\mathbf{A}$ (resp. $\mathbf{B}$) consists of the coefficients of a_i (resp. b_i), for all $0 \leq i < d$. We let $\mathtt{Toep}(\mathsf{sk})$ denote the matrix whose i-th row consist of the coefficients of $X^i \cdot s \in \mathcal{R}$, for all $0 \leq i < d$. Note that the matrix $\mathtt{Toep}(\mathsf{sk})$ is structured, whereas $\mathbf{A}$ and $\mathbf{B}$ do not have a particular structure. Now, multiplying by the plaintext matrix $\mathbf{U} \in \mathbb{R}^{d \times d}$ on both sides of (1) gives:

$$(\mathbf{U} \cdot \mathbf{A}) \cdot \mathtt{Toep}(\mathsf{sk}) + (\mathbf{U} \cdot \mathbf{B}) \;\approx\; \mathbf{U} \cdot \mathbf{M} \mod q \;\;. \tag{2}$$

Note that the error term hidden in the $\approx$ symbol has also been multiplied by $\mathbf{U}$, which should be taken into account when setting parameters. Now, note that Equation (2) is of the same form as (1). Defining a'_i (resp. b'_i) as the element of $\mathcal{R}_q$ whose coefficients correspond to the i-th row of $\mathbf{U} \cdot \mathbf{A}$ (resp. $\mathbf{U} \cdot \mathbf{B}$) for all $0 \leq i < d$, we obtain that the (a_i, b_i)'s are an encryption of $\mathbf{U} \cdot \mathbf{M}$. Overall, the PC-MM $\mathbf{U} \cdot \mathbf{M}$ reduces to the PP-MM's $\mathbf{U} \cdot \mathbf{A}$ and $\mathbf{U} \cdot \mathbf{B}$ (modulo q).

When $d = kN$ for some integer $k \geq 1$, the approach generalizes as follows. Each row is encrypted using several RLWE ciphertexts (in total, there are d^2/N of them). This leads to an equation of the form:

$$\mathbf{A} \cdot (\mathbf{I} \otimes \mathtt{Toep}(\mathsf{sk})) + \mathbf{B} \;\approx\; \mathbf{M} \mod q \ .$$

By multiplying on the left by $\mathbf{U}$, one again obtains a PC-MM algorithm that consists of two PP-MM's modulo q.

Shared-a RLWE ciphertexts and large dimensions. To decrease the cost of large-dimensional PC-MM, we use multi-secret RLWE ciphertexts sharing their a-parts. The matrix $\mathbf{M}$ is provided by d^2/N multi-secret RLWE ciphertexts $(a_i, b_{i,j})_{0 \leq i < d, 0 \leq j < k}$. Note that the number of a_i's is only d. We have, over $\mathcal{R}_q$:

$$\forall i, j : a_i \cdot \mathsf{sk}_j + b_{i,j} \;\approx\; \sum_t m_{i,Nj+t} X^t \ ,$$

where the sk_j's in $\mathcal{R}$ are the secret keys. In matrix form, this gives:

$$\mathbf{A} \cdot [\mathtt{Toep}(s_0) | \cdots | \mathtt{Toep}(s_{k-1})] + \mathbf{B} \;\approx\; \mathbf{M} \mod q \ , \tag{3}$$

where $\mathbf{A}$ (resp. $\mathbf{B}$) consists of the coefficients of a_i (resp. $b_{i,j}$) and is of size $d \times N$ (resp. $d \times d$). As in RLWE case, we multiply $\mathbf{U}$ on both sides of the equation:

$$(\mathbf{U} \cdot \mathbf{A}) \cdot [\mathtt{Toep}(\mathsf{sk}_0) | \cdots | \mathtt{Toep}(\mathsf{sk}_{k-1})] + (\mathbf{U} \cdot \mathbf{B}) \;\approx\; (\mathbf{U} \cdot \mathbf{M}) \mod q \ ,$$

Since it has the same form as the original matrix equation, we can view it as d^2/N multi-secret RLWE ciphertexts $(a'_i, b'_{i,j})$ with respect to secret keys $\mathsf{sk}_0, \cdots, \mathsf{sk}_{k-1}$. This gives us a reduction from PC-MM (with multi-secret RLWE) of $\mathbf{U}$ and $\mathbf{M}$ to the PP-MM's of $\mathbf{U}$ with $\mathbf{A}$ and $\mathbf{U}$ with $\mathbf{B}$. Importantly, the number of columns of $\mathbf{A}$ is smaller than that of $\mathbf{M}$, by a factor k, giving a significant saving compared to [26] for large d.

MLWE ciphertexts and small dimensions. For smaller matrices, with $dk = N$ for some integer k, RLWE ciphertexts contain several rows of the input ciphertext matrix. Operating directly on these ciphertexts for the PC-MM seems difficult, as it is likely to involve expensive key-switching operations. To handle this setup, we rely on MLWE ciphertexts. The matrix $\mathbf{M}$ is provided by d MLWE ciphertexts $(\mathbf{a}_i, b_i)_{0 \leq i < d}$, i.e., we have, over $\mathbb{Z}_q[X]/(X^d + 1)$:

$$\forall i, j : \langle \mathbf{a}_i, \mathbf{sk} \rangle + b_i \;\approx\; \sum_t m_{i,j} X^j \ ,$$

where $\mathbf{sk} = (\mathsf{sk}_0, \cdots, \mathsf{sk}_{k-1}) \in (\mathbb{Z}_q[X]/(X^d+1))^k$ is the secret key. We can rewrite the above in matrix form as follows:

$$\mathbf{A} \cdot \begin{bmatrix} \mathtt{Toep}(\mathsf{sk}_0) \\ \hline \vdots \\ \hline \mathtt{Toep}(\mathsf{sk}_{k-1}) \end{bmatrix} + \mathbf{B} \;\approx\; \mathbf{M} \mod q \;,$$

where $\mathbf{A}$ (resp. $\mathbf{B}$) consists of the coefficients of a_i (resp. b_i) and is of size $d \times N$ (resp. $d \times d$). Again, multiplying both sides of the equation by $\mathbf{U}$ gives a PC-MM algorithm. The result corresponds to d MLWE ciphertexts (a'_i, b'_i).

Shared-a ciphertexts and precomputation. Allowing precomputation, we can even reduce PC-MM to a single PP-MM modulo q during the online phase. For ease of discussion, assume that $d = N$. We can rewrite the multi-secret ciphertexts (a, b_i), which encrypt the rows of a matrix $\mathbf{M}$ under secret keys sk_i, in matrix form, as follows:

$$\mathbf{S} \cdot \mathtt{Toep}(a) + \mathbf{B} \;\approx\; \mathbf{M} \mod q \;, \tag{4}$$

where the i-th row of $\mathbf{S}$ (resp. $\mathbf{B}$ and $\mathtt{Toep}(a)$) consists of the coefficients of sk_i (resp. b_i and $x^i \cdot a \in \mathcal{R}_q$), for all $0 \le i < d$. Note that this is a different form of matrix equation from Equation (3). By multiplying $\mathbf{U}$ to both sides, we get:

$$(\mathbf{U} \cdot \mathbf{S}) \cdot \mathtt{Toep}(a) + (\mathbf{U} \cdot \mathbf{B}) \;\approx\; (\mathbf{U} \cdot \mathbf{M}) \mod q \;. \tag{5}$$

This has the same form as Equation (4), and hence we can view it as multi-secret RLWE ciphertexts under secret key sk'_i, where sk'_i consists of the i-th row of $\mathbf{S}' = \mathbf{U} \cdot \mathbf{S}$. If the computation continues with a specific secret key, e.g., sk, we can switch the keys from sk'_i to sk after PC-MM, using CKKS key-switching. The switching keys can be precomputed, and the key-switching cost $\widetilde{O}(dN)$ is relatively small compared to the one of the PP-MM.

For matrices of size $d > N$ or $d < N$, we describe analogous approaches in Section 4.3. For $d > N$, we use multi-secret RLWE, and for $d < N$, we use multi-secret MLWE. In all cases, in terms of online cost, we reduce one PC-MM to one PP-MM.

PC-MM with a single floating-point PP-MM. So far, all PP-MM's are matrix products over $\mathbb{Z}_q$ for some integer q. Modular PP-MM is typically slower than floating-point PP-MM, as it does not directly benefit from the high-performance of numerical BLAS libraries. However, for approximate homomorphic encryption, the least significant bits of the b-parts of the RLWE ciphertexts contain numerical and RLWE errors that are not relevant. Thus, to multiply $\mathbf{U}$ and $\mathbf{B}$, we use floating-point PP-MM on the most significant bits of $\mathbf{B}$, instead of computing PP-MM modulo q. We provide an error analysis in Section 5.

We can apply this optimization to all suggested PC-MM algorithms. It is most effective for large-dimensional matrices (as the cost of computing $\mathbf{UB}$ is then higher than that of computing $\mathbf{UA}$) and for the algorithms using precomputation. In particular, for PC-MM with precomputation, this optimization provides a reduction to a single floating-point PP-MM.

Batch bootstrapping. CKKS bootstrapping [9] takes as input a ciphertext with a small modulus, and transforms it in a ciphertext with a much higher modulus that decrypts to the same message (up to numerical error). This enables to perform further homomorphic operations on the ciphertext. Assume we aim at simultaneously bootstrapping several ciphertexts. This is a typical scenario after a PC-MM, as the output matrix may not fit into a single RLWE ciphertext, but can occur in other large-scale computations.

The shared-a format proves very useful for batch-bootstrapping. First, we provide conversion algorithms from shared-secret RLWE ciphertexts to shared-a RLWE ciphertexts, so that the format can be used within CKKS. Second, we observe that this format enjoys a number of homomorphic operations. For example, consider two ciphertexts (a, b_1) and (a, b_2) such that $a \cdot \mathsf{sk}_1 + b_1 = m_1$ and $a \cdot \mathsf{sk}_2 + b_2 = m_2$. Then, we have

$$a \cdot (\mathsf{sk}_1 + \mathsf{sk}_2) + (b_1 + b_2) = (m_1 + m_2) \ .$$

The resulting ciphertext (a, b_1+b_2) still has the same a-part and decrypts to m_1+m_2. Similarly, the format is compatible with multiplication by a plaintext:

$$a \cdot \mathsf{sk} + b = m \ \Rightarrow \ a \cdot (u \cdot \mathsf{sk}) + (u \cdot b) = (u \cdot m),$$

for any $u \in \mathcal{R}$. Note that multiplication between two ciphertexts does not enjoy such a property. Other operations that are compatible with the format include rescaling (decreasing the current modulus) and key-switching (hence allowing for so-called rotations and conjugation).

Now, recall that CKKS bootstrapping consists of several components: S2C, ModRaise, C2Sand EvalMod. As ModRaise is purely semantic, it is compatible with the shared-a format. More importantly, S2C and C2S rely only on additions, multiplications by plaintexts, rotations and conjugations. They are hence compatible with the shared-a format. As a result, all computations involving the a-parts of the ciphertexts in S2C and C2S can be amortized across a batch of ciphertexts.

MaMBo: fused PC-MM and batch bootstrapping. Finally, we explain how to combine PC-MM with batch bootstrapping. PC-MM is often quite expensive, and it is often preferable to place it at the smallest possible modulus. In MaMBo, we propose to first convert the shared-secret input ciphertexts into shared-a ciphertexts. Then we run batch-S2C such that the output level still enables a matrix multiplication. One can then execute one of the PC-MM algorithms described above, on an input which has already been converted to coefficient-encoding. After ModRaise, we run batch-C2S and finally convert from shared-a ciphertexts to shared-secret ciphertexts before EvalMod. This approach can be further improved by taking advantage of subrings to decrease the ring dimension. When $d \geq N$, our algorithms also work on slot-encoded input; for the sake of completeness, in that case, we also discuss other ways to combine PC-MM and bootstrapping.

2 Preliminaries

As our paper is notation-heavy, we provide a table of the main notations and functions in Appendix E.

Vectors (resp. matrices) are denoted in bold lower-case (resp. upper-case) letters. Unless explicitly stated otherwise, vectors are column vectors. We let $\langle \cdot, \cdot \rangle$ denote the formal dot product of two vectors: for any ring R and dimension k, for $\mathbf{x} = (x_i)_i, \mathbf{y} = (y_i)_i \in R^k$, we write $\langle x, y \rangle = \sum_i x_i y_i$. For a matrix $\mathbf{A}$, the notation $a_{i,[k,\ell)}$ refers to the entries $a_{i,j}$ for $k \leq j < \ell$. For N a power of two and $q \geq 2$, we define $\mathcal{R}_N = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_{q,N} = \mathbb{Z}_q[X]/(X^N + 1)$. We skip the index N when it is clear from the context.

For a real number x, we let $\lfloor x \rceil$ denote the integer part of $x + 1/2$. The notation log refers to the base-2 logarithm. We let ω refer to the complexity exponent of square matrix multiplication and assume that $\omega > 2$. As in Python, in our pseudo-codes, for loops for some variable i from a to b means that i takes values $i = a, a + 1, \ldots, b - 1$.

2.1 RLWE and MLWE ciphertext formats

For N a power of 2, and $Q \geq 2$ an integer, a $\mathsf{RLWE}_{Q,N}$ ciphertext for a plaintext $m \in \mathcal{R}_{Q,N}$ under a secret key $\mathsf{sk} \in \mathcal{R}_N$ is a pair $(a, b) \in \mathcal{R}_{Q,N}^2$ such that $a \cdot \mathsf{sk} + b = m$. In this work, we consider plaintexts m that are:

- small, i.e., have coefficients with small absolute values compared to Q;
- approximate, i.e., m could as well be $m + e$ for a small e.

Such ciphertexts can be generated in a symmetric manner, if the encryptor knows sk, or in an asymmetric manner if the encryptor knows encryptions of 0. The ciphertexts are indistiguishable from uniform for any adversary that does not know sk, under the RLWE hardness assumption [33, 27].

The MLWE ciphertext format [6, 25] is the following generalization. Let $k \geq 1$. An $\mathsf{MLWE}_{Q,N}^{(k)}$ ciphertext for a plaintext $m \in \mathcal{R}_{Q,N}$ under a secret key $\mathbf{sk} \in \mathcal{R}_N^k$ is a pair $(\mathbf{a}, b) \in \mathcal{R}_{Q,N}^k \times \mathcal{R}_{Q,N}$ such that $\langle \mathbf{a}, \mathbf{sk} \rangle + b = m$. Here k, N and Q are respectively referred to as (module) rank, (ring) degree and modulus. Note that $\mathsf{MLWE}_{Q,N}^{(1)}$ and $\mathsf{RLWE}_{Q,N}$ coincide.

2.2 The CKKS scheme

CKKS [10] is an RLWE-based FHE scheme that performs approximate computations on complex numbers. The CKKS plaintexts are elements of $\mathcal{R}_N$, which can encode complex vectors as described below.

Encoding. Let $\zeta_j = \exp(2i\pi 5^j / 2N)$ for $0 \leq j < N/2$ be an ordered set of pairwise non-conjugate primitive $2N$-th roots of unity. The map $P \mapsto (P(\zeta_j))_j$ is an isomorphism between $\mathcal{R}$ and $\mathbb{C}^{N/2}$, which by analogy to the discrete Fourier

transform, shall be denoted by DFT. Similarly, we will let iDFT denote its inverse. Given an integer Δ, the CKKS *slot encoding* process is defined as:

$$m = \mathsf{Ecd}_{\mathrm{slot}}(z) = \lfloor \Delta \cdot \mathrm{iDFT}(z) \rceil \in \mathcal{R} \ .$$

Conversely, given a plaintext m, we say that its j-th slot contains z_j if $m(\zeta_j) = z_j$. Given as input $m \in \mathcal{R}$, the decoding process then returns

$$z = \mathsf{Dcd}_{\mathrm{slot}}(m) = \frac{1}{\Delta}\mathrm{DFT}(m) \in \mathbb{C}^{N/2} \ .$$

We have $\mathsf{Dcd}(\mathsf{Ecd}_{\mathrm{slot}}(z)) \approx z$, the error being of the order of $\sqrt{N}/\Delta$.

We also define *coefficient encoding* for *real* messages. If $z \in \mathbb{R}^n$, we define

$$\mathsf{Ecd}_{\mathrm{coeff}}(z) = \sum_{i=0}^{N-1} \lfloor \Delta \cdot z_i \rceil X^i \in \mathcal{R}.$$

It is typically only used for internal operations, mostly in the bootstrapping process, but will play an important role in this paper.

Basic functionalities. CKKS provides the following elementary operations on keys and ciphertexts:

- KeyGen. Given a security parameter λ, $\mathsf{KeyGen}(1^\lambda)$ returns public key $\mathsf{pk} = (a, b) \in \mathcal{R}_Q^2$ for some Q, and a secret key $\mathsf{sk} \in \mathcal{R}$;
- Enc. For a plaintext m (which is obtained by an encoding function as explained above), $\mathsf{Enc}(\mathsf{pk}, m)$ returns a RLWE-format ciphertext (a, b) such that $a \cdot \mathsf{sk} + b = m + e$, where e has small magnitude coefficients.
- Dec. Given a ciphertext (a, b), $\mathsf{Dec}(\mathsf{sk}, (a, b))$ returns $a \cdot \mathsf{sk} + b$.
- SWKGen. Given two integers P, Q and two keys $\mathsf{sk}, \mathsf{sk}' \in \mathcal{R}_N$, SWKGen returns a switching key $\mathsf{swk}_{Q,P,\mathsf{sk}\to\mathsf{sk}'}$ from key sk to key sk' for ciphertexts modulo Q and with auxiliary integer P. It is of the form:

  $$\mathsf{swk}_{Q,P,s\to s'} = (a_{\mathsf{swk}}, b_{\mathsf{swk}}) = (a, -a \cdot \mathsf{sk}' + e + P \cdot \mathsf{sk}) \in R_{PQ}^2 \ ,$$

  where $e \in R_N$ has small-magnitude coefficients.[5]
- $\mathsf{KeySwitch}$. Given as input a mod-Q ciphertext ct for a message m under sk, $\mathsf{KeySwitch}(\mathsf{swk}_{Q,P,\mathsf{sk}\to\mathsf{sk}'}, \mathsf{ct})$ returns a mod-Q ciphertext ct' for m under sk'.

Homomorphic operations. We now describe a subset of the homomorphic operations provided by CKKS.

- Add. Given two ciphertexts ct and ct' for the same modulus Q, Add computes and returns $\mathsf{ct}'' = \mathsf{Add}(\mathsf{ct}, \mathsf{ct}') \approx \mathsf{ct} + \mathsf{ct}'$. If $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \approx m$ and $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') = m'$, then we have $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}'') \approx m + m'$.

[5] Switching keys can also be defined with more ring elements when using a so-called gadget rank larger than 1. Our techniques also carry over to that setup, but we restrict ourselves to switching keys as above for the sake of simplicity.

- Rot. Given ct such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \approx \mathsf{Ecd}_{\mathrm{slot}}(x_0, \ldots, x_{N/2-1})$ and an index i, Rot returns ct' such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') \approx \mathsf{Ecd}_{\mathrm{slot}}(x_i, \ldots, x_{N/2-1}, x_0, \ldots, x_{i-1})$. This requires the rotation key rk_i, which is a switching key from $\mathsf{sk}(X^{5^i})$ to sk.
- Conj. On input ct such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \approx \mathsf{Ecd}_{\mathrm{slot}}(x_0, \ldots, x_{N/2} - 1)$, Conj returns ct' such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') \approx \mathsf{Ecd}_{\mathrm{slot}}(\overline{x}_0, \ldots, \overline{x}_{N/2}-1)$, where, for $x \in \mathbb{C}$, where $\overline{x}$ stands for the complex conjugate of x. This requires the conjugation key ck, which is a switching key from $\mathsf{sk}(X^{-1})$ to sk.

Before defining homomorphic multiplication, we first discuss levels and rescaling. As $\mathsf{Ecd}(x) \approx \Delta \cdot \mathrm{iDFT}(x)$, the multiplication of two encodings leads to a result that is scaled by a factor Δ^2:

$$\mathsf{Ecd}(x) \cdot \mathsf{Ecd}(x') \;\approx\; (\Delta \cdot \mathrm{iDFT}(x)) \cdot (\Delta \cdot \mathrm{iDFT}(x')) \;\approx\; \Delta^2 \cdot \mathrm{iDFT}(x \odot x') \;,$$

where $\odot$ refers to the componentwise product. In order to restore a Δ-scaled encoding, one can divide this result by Δ. At the ciphertext level, this is approximately achieved by means of the following function:

- $\mathsf{Rescale}$. Given $q > q'$ and a ciphertext ct modulo q, $\mathsf{Rescale}$ computes and returns $\lfloor (q'/q) \cdot \mathsf{ct} \rceil \in \mathcal{R}^2_{q'}$. If ct decrypts to m, then so does ct' (up to some error).

Note that the ciphertext modulus has changed from q to q': this is accounted for by the notion of *level*. Let q_0 be a prime, and $q_1, \ldots, q_\ell$ be primes $\approx \Delta$. We define $Q_j = q_0 q_1 \ldots q_j$. A ciphertext at level ℓ is an element of $\mathcal{R}^2_{Q_\ell}$; after a multiplicative operation, rescaling moves ct in $\mathcal{R}^2_{Q_{\ell-1}}$: the level of ct becomes $\ell - 1$. Note that homomorphic addition and multiplication require the two input ciphertexts to be at the same level (which can be achieved by using the $\mathsf{Rescale}$ function). The output of addition remains at the same level, whereas the output of multiplication is one level less. We let Q_{top} denote the largest Q_i – for a given N, this value is determined so that $\mathsf{RLWE}_{Q_{\mathsf{top}},N}$ remains sufficiently hard to fit the desired security level.

We restrict ourselves to the description of plaintext-ciphertext multiplication, as we will not use of ciphertext-ciphertext multiplication.

- PCMult. Given $\alpha \in \mathbb{C}^{N/2}$ and $\mathsf{ct} \in \mathcal{R}^2_{Q_k,N}$, PCMult computes and returns $\mathsf{ct}' = \mathsf{Rescale}_{Q_k,Q_{k-1}}(\mathsf{Ecd}_{\mathrm{slot}}(\alpha) \cdot \mathsf{ct}) \in \mathcal{R}_{Q_{k-1},N}$. If $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \approx \mathsf{Ecd}_{\mathrm{slot}}((m_i)_i)$, then we have $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') \approx \mathsf{Ecd}_{\mathrm{slot}}((\alpha_i m_i)_i)$.

When $\alpha \in \mathbb{R}$ is a scalar, we can also define PCMult for coefficient encoding, which on input a ciphertext encrypting $\mathsf{Ecd}_{\mathrm{coeff}}(m)$ returns a ciphertext encrypting $\mathsf{Ecd}_{\mathrm{coeff}}(\alpha \cdot m)$.

Bootstrapping. When a ciphertext ct reaches level 0, i.e., $\mathsf{ct} \in \mathcal{R}_{Q_0}$, its multiplicative level must be upgraded in order to allow for further multiplications. This is achieved via a process called *bootstrapping*.

A significant number of important technical details being left aside, the main steps of this process are the following:

- $\mathsf{S2C}$ (for *slot-to-coeffs*) is a linear step which homomorphically changes the internal encoding of the message, from $\mathsf{ct} = \mathsf{Enc}(\mathsf{Ecd}_{\mathrm{slot}}(m))$ to $(a, b) = \mathsf{Enc}(\mathsf{Ecd}_{\mathrm{coeff}}(m))$;
- $\mathsf{ModRaise}$ lifts the ciphertext (a, b) from $\mathcal{R}_{Q_0}$ to $\mathcal{R}_{Q_{\mathsf{top}}}$ while preserving an identity $as+b = \mathsf{Ecd}_{\mathrm{coeff}}(m)+Q_0 \cdot I \mod Q_{\mathsf{top}}$, where I is a small (unknown) integer;
- $\mathsf{C2S}$ (for *coeffs-to-slots*) is the inverse operation of $\mathsf{S2C}$ and recovers ct' encrypting $\mathsf{Ecd}_{\mathrm{slot}}(m) + Q_0 I$;
- $\mathsf{EvalMod}$ homomorphically removes the $Q_0 \cdot I$ part to recover a ciphertext encrypting $\mathsf{Ecd}_{\mathrm{slot}}(m)$ in $\mathcal{R}_Q$ for some intermediate modulus Q.

The CKKS bootstrapping chain comes in the following two flavours.

- $\mathsf{S2C}$-first bootstrapping follows the steps in the order above. $\mathsf{S2C}$ is then executed at a low level, with a lower cost. $\mathsf{S2C}$-first bootstrapping should be started at a sufficiently high level to allow the homomorphic evaluation of the $\mathsf{S2C}$ function before reaching level 0 and re-increasing the modulus.
- $\mathsf{C2S}$-first bootstrapping starts with $\mathsf{ModRaise}$ which first raises the modulus to Q_{top}. It can thus be started at level 0. After raising the ciphertext to Q_{top}, $\mathsf{C2S}$, $\mathsf{EvalMod}$ and $\mathsf{S2C}$ are successively executed before the data can be further processed. The whole bootstrapping chain is executed at high moduli, for a higher cost.

In $\mathsf{S2C}$-first bootstrapping, we call $\mathsf{HalfBTS}$ the part of the bootstrapping process that follows $\mathsf{S2C}$. We let ℓ_{S2C} denote the number of levels used for $\mathsf{S2C}$. In practice, the choice of $\ell_{\mathsf{S2C}} = 3$ is common.

2.3 From rings to subrings and modules, and back

Throughout the end of this section, the integer N' is a power of 2 dividing N, and we let $k = N/N'$. We shall consider four operations: moving from $\mathsf{RLWE}_{Q,N}$ to $\mathsf{RLWE}_{Q,N'}$ and back; and moving from $\mathsf{RLWE}_{Q,N}$ to $\mathsf{MLWE}^{N/N'}_{Q,N'}$ and back. When Q is too large for the former to preserve security, we shall use the latter as a substitute. We only give the specifications of the operations, We provide details on how they are implemented in Appendix A.

Ring-switching. Ring-switching is a technique introduced in [6] and further developed in [19]. The goal is to decompose a large-ring ciphertext into several small-ring ciphertexts.

- $\mathsf{RingSwitch}$. On input $\mathsf{ct} = (a, b) \in \mathcal{R}^2_{Q,N}$ and $\mathsf{swk}_{Q,P,\mathsf{sk}\to\mathsf{sk}'}$, $\mathsf{RingSwitch}$ returns $(a_i, b_i) \in \mathcal{R}^2_{Q,N'}$ for $0 \le i < k$ such that, when viewed together, the underlying plaintexts correspond to the plaintext underlying (a, b).

Conversely, starting from k ciphertexts $\mathsf{ct}_i \in \mathcal{R}_{Q,N'}$ for a common key $\mathsf{sk}' \in \mathcal{R}_{N'}$ and a switching key $\mathsf{swk}_{P,Q,\mathsf{sk}'\to\mathsf{sk}}$ from sk' to sk, we can key-switch the ct_i's from sk' to sk, to restore a $\mathsf{RLWE}_{Q,N}$ ciphertext with key in $\mathcal{R}_N$.

From RLWE to MLWE. When PQ is too large, providing the required switching key would impair the security. In this case, we can switch from $\mathsf{RLWE}_{Q,N}$ to $\mathsf{MLWE}^k_{Q,N'}$ as a secure alternative; the latter does not require key switching.

- $\mathsf{ModDecomp}$. On input $\mathsf{ct} = (a, b)$, $\mathsf{ModDecomp}$ returns $(\mathbf{a}_i, b_i) \in \mathcal{R}^{k+1}_{Q,N'}$ for $0 \leq i < k$, such that, when viewed together, the underlying plaintexts correspond to the plaintext underlying (a, b).

Here a and sk play symmetric roles, and we can equivalently obtain a decomposition in k ciphertexts with common a part, but distinct secret keys; we call $\mathsf{ShModDecomp}$ the corresponding procedure.

From MLWE to RLWE. Going back from $k = N/N'$ $\mathsf{MLWE}^k_{Q,N'}$ ciphertexts to one $\mathsf{RLWE}_{N,Q}$ ciphertext is an instance of the *ring-packing* problem. This problem has been extensively studied (see, e.g., [11, 4, 8, 2]) in the context of packing multiple LWE ciphertexts into an RLWE ciphertext. The more general task of packing $\mathsf{MLWE}_{Q,N'}$ ciphertexts into an $\mathsf{MLWE}_{Q,N}$ ciphertext is addressed in [2].

- $\mathsf{ModPack}$. On input $(\mathbf{a}_i, b_i) \in \mathcal{R}^{k+1}_{Q,N'}$ for $0 \leq i < k$ for a common key $\mathbf{sk} \in \mathcal{R}^k_{N'}$, $\mathsf{ModDecomp}$ returns a ciphertext $(a, b) \in \mathcal{R}^2_{Q,N}$ whose underlying plaintext corresponds to the tuple of plaintexts underlying the $(\mathbf{a}_i, b_i)$'s. This requires switching keys.

We will also use the $\mathsf{ModKeySwitch}$ algorithm from [2], which is the MLWE version of $\mathsf{KeySwitch}$.

3 Multiple-secret RLWE with shared a-part

Recall that a RLWE ciphertext for a secret $\mathsf{sk} \in \mathcal{R}$ and a plaintext $m \in \mathcal{R}$ is a pair $(a, b) \in \mathcal{R}^2_Q$ such that $a \cdot \mathsf{sk} + b \approx m \bmod Q$. We refer to a (resp. b) as the a-part (resp. b-part). A RLWE ciphertext hence consumes two elements of $\mathcal{R}_Q$ to represent a single ring element m. This factor 2 also impacts the performance of homomorphic operations.

A common approach is to seed the a-part, i.e., implicitly represent a by a much more compact $seed \in \{0,1\}^{128}$ as $a = H(seed) \in \mathcal{R}_Q$, where H is an extendable output function (such as SHAKE) modeled as a random oracle. This works well when the ciphertext (a, b) is the output of the encryption algorithm, in the symmetric setting (i.e., when the encryptor knows the secret key). After the ciphertext has been computed upon, the a-part becomes a combination of several a-parts and b-parts of various ciphertexts and components of the evaluation key, and it cannot be represented in seeded form anymore. We place ourselves in this context.

3.1 RLWE ciphertexts with shared a-part

Consider n RLWE ciphertexts $\mathsf{ct}_i = (a_i, b_i) \in \mathcal{R}_Q^2$, $0 \leq i < n$, under a common key $\mathsf{sk} \in \mathcal{R}$ and for plaintexts $m_i \in \mathcal{R}$:

$$\forall i, 0 \leq i < n : a_i \cdot \mathsf{sk} + b_i \approx m_i \bmod Q \ .$$

We consider a different format for encoding those n plaintexts, involving n secret keys $\mathsf{sk}_0, \ldots, \mathsf{sk}_{n-1} \in R$. It consists of $n+1$ ring elements a and $b_0, \ldots, b_{n-1}$ such that

$$\forall i, 0 \leq i < n : a \cdot \mathsf{sk}_i + b_i \approx m_i \ .$$

This can be interpreted as n RLWE ciphertexts $\mathsf{ct}_i = (a_i, b_i)$ under keys sk_i and that satisfy $a_i = a$ for all i. In short, they share the a-part. Overall, to store n plaintexts, one only needs $n + 1$ elements of R_q instead of $2n$.

3.2 Forward format conversion

Assume that we have n ciphertexts $\mathsf{ct}_i = (a_i, b_i) \in \mathcal{R}_Q^2$ such that $a_i \cdot \mathsf{sk} + b_i \approx m_i$ for some shared secret key sk and plaintexts m_i (for all i with $0 \leq i < n$). We aim at converting them to n ciphertexts $\mathsf{ct}'_i = (a', b'_i) \in \mathcal{R}_Q^2$ such that $a' \cdot \mathsf{sk}'_i + b'_i \approx m_i$ (for all i with $0 \leq i < n$), which share their a-parts. In this context, the secret keys $\mathsf{sk}, \mathsf{sk}'_0, \ldots, \mathsf{sk}'_{n-1}$ are all sampled during key generation. We assume the key generation algorithm also outputs a format-switching key:

$$\mathsf{fmt\text{-}swk}_{Q,P,\mathsf{sk}\to\{\mathsf{sk}'_i\}_{0\leq i<n}} = (\mathbf{a}_{\mathsf{fmt\text{-}swk}}, \mathbf{B}_{\mathsf{fmt\text{-}swk}}) \in \mathcal{R}_{PQ}^n \times \mathcal{R}_{PQ}^{n\times n} \ ,$$

for some auxiliary integer P with $\mathbf{a}_{\mathsf{fmt\text{-}swk}}$ sampled uniformly in $\mathcal{R}_{PQ}^n$, and

$$\mathbf{B}_{\mathsf{fmt\text{-}swk}} = -\mathbf{a}_{\mathsf{fmt\text{-}swk}} \cdot \left(\mathsf{sk}'_0, \ldots, \mathsf{sk}'_{n-1}\right) + \mathbf{E} + P \cdot \mathsf{sk} \cdot \mathbf{I}_n \bmod PQ \ , \qquad (6)$$

where $\mathbf{E} \in \mathcal{R}^{n\times n}$ has small-magnitude coefficients, and $\mathbf{a}_{\mathsf{fmt\text{-}swk}} \cdot \left(\mathsf{sk}'_0, \ldots, \mathsf{sk}'_{n-1}\right)$ is the multiplication of a column matrix and a row matrix, resulting in a $n \times n$ matrix. Note that $\mathbf{a}_{\mathsf{fmt\text{-}swk}}$ can be seeded, to reduce memory consumption. Nevertheless, the matrix $\mathbf{B}_{\mathsf{fmt\text{-}swk}}$ still contains n^2 elements of $\mathcal{R}_{PQ}$.

By a standard hybrid argument (see [31, Le. 6.2]), one can show that the format conversion key $\mathsf{fmt\text{-}swk}_{Q,P,\mathsf{sk}\to\{\mathsf{sk}'_i\}_{0\leq i<n}}$ is computationally indistinguishable from uniform.

We can now introduce the format key switching procedure.

- $\mathsf{FmtSwitch}$. Given as input ciphertexts $\{(a_i, b_i)\}_{0\leq i<n}$ for a secret key sk, the $\mathsf{FmtSwitch}$ algorithm uses the format switching key $(\mathbf{a}_{\mathsf{fmt\text{-}swk}}, \mathbf{B}_{\mathsf{fmt\text{-}swk}})$ to compute and return:

$$a' = \left\lfloor \frac{1}{P} \Big(\langle (a_0, \ldots, a_{n-1})^T, \mathbf{a}_{\mathsf{fmt\text{-}swk}} \rangle \bmod PQ \Big) \right\rceil \ ,$$

$$(b'_0, \ldots, b'_{n-1}) = (b_0, \ldots, b_{n-1}) + \left\lfloor \frac{1}{P} \Big((a_0, \ldots, a_{n-1}) \cdot \mathbf{B}_{\mathsf{fmt\text{-}swk}} \bmod PQ \Big) \right\rceil \ .$$

Note that the cost of $\mathsf{FmtSwitch}$ is dominated by $O(n^2)$ multiplications and additions in $\mathcal{R}_{PQ}$.

Lemma 1. *Using the notations above, assume that* $a_i \cdot \mathsf{sk} + b_i \approx m_i \bmod Q$ *for all* $0 \le i < n$. *If* $P \ge Q$, *then* $a' \cdot \mathsf{sk}'_i + b'_i \approx m_i \bmod Q$ *for all* $0 \le i < n$.

Proof. Let $0 \le i < n$. There exist $e_a, e_b \in \mathbb{R}[X]/(X^N+1)$ with coefficients in $[-1/2, 1/2]$ and $k_a, k_b \in \mathcal{R}$ such that, over $\mathbb{R}[X]/(X^N+1)$:

$$a' \cdot \mathsf{sk}'_i = \frac{1}{P}\langle (a_0, \ldots, a_{n-1})^T, \mathbf{a}_{\mathsf{fmt\text{-}swk}} \rangle \cdot \mathsf{sk}'_i + e_a \cdot \mathsf{sk}'_i + k_a Q \ ,$$
$$b'_i = b_i + \frac{1}{P}(a_0, \ldots, a_{n-1}) \cdot \mathbf{B}_{\mathsf{fmt\text{-}swk},i} + e_b + k_b Q \ ,$$

where $\mathbf{B}_{\mathsf{fmt\text{-}swk},i}$ refers to the i-th column of $\mathbf{B}$. Taking the i-th column of Equation (6), we obtain that there exists $\mu \in \mathcal{R}$, such that the following holds over $\mathcal{R}$:

$$\mathbf{a}_{\mathsf{fmt\text{-}swk}} \cdot \mathsf{sk}'_i + \mathbf{B}_{\mathsf{fmt\text{-}swk},i} = \mathbf{e}_i + P \cdot (0, \ldots, 0, \mathsf{sk}, 0, \ldots, 0)^T + \mu P Q \ ,$$

where $\mathbf{e}_i$ refers to the i-th column of $\mathbf{E}$ and sk is in the i-th coefficient of $(0, \ldots, 0, \mathsf{sk}, 0, \ldots, 0)^T$. Combining the last three equations, we obtain:

$$\begin{aligned} a' \cdot \mathsf{sk}'_i + b'_i &= a_i \cdot \mathsf{sk} + b_i + (k_a + k_b + \mu) Q \\ &\quad + \left(\frac{1}{P}\langle (a_0, \ldots, a_{n-1})^T, \mathbf{e}_i \rangle + e_a \cdot \mathsf{sk}'_i + e_b \right) \\ &\approx m_i + (k_a + k_b + \mu) Q \ . \end{aligned}$$

Here, we used the fact that the a_i's have coefficients in $[-Q/2, Q/2]$ and $P \ge Q$, implying that the ring element $\frac{1}{P}\langle (a_0, \ldots, a_{n-1})^T, \mathbf{e}_i \rangle$ has small-magnitude coefficients. Reducing modulo Q gives the result. □

3.3 Improved forward format conversion

A drawback of the format conversion approach described above is the quadratic growth of the size of the format conversion key and the quadratic cost, as a function of n. We propose an alternative recursive approach: first, each pair of ciphertexts is transformed so that they share their a-parts (i.e., there are only $n/2$ distinct a-parts); then each pair of pair of ciphertexts is transformed so that they share their a-parts (i.e., there are only $n/4$ distinct a-parts), etc.

For the sake of simplicity, assume that $n = 2^\kappa$ for some integer $\kappa \ge 1$. We define $\mathbf{sk}^{(0)} = \mathsf{sk}$ and $\mathbf{sk}^{(\kappa)} = (\mathsf{sk}'_0, \ldots, \mathsf{sk}'_{2^\kappa - 1})$. For $\ell \in [1, \kappa - 1]$, we sample secret keys $\mathsf{sk}^{(\ell)}_0, \ldots, \mathsf{sk}^{(\ell)}_{2^\ell - 1}$ and define $\mathbf{sk}^{(\ell)}$ as their concatenation (we could alternatively define $\mathbf{sk}^{(\ell)}$ as the first 2^ℓ elements of $\mathbf{sk}^{(\kappa)}$). We define the recursive format conversion key as follows:

$$\widetilde{\mathsf{fmt\text{-}swk}}_{Q,P,\mathsf{sk} \to \{\mathsf{sk}'_i\}_{0 \le i < n}} = \{(\mathbf{a}^{(\ell)}_{\mathsf{fmt\text{-}swk}}, \mathbf{B}^{(\ell)}_{\mathsf{fmt\text{-}swk}})\}_{\ell \in [1,\kappa]} \ ,$$

for some auxiliary integer P, with $\mathbf{a}^{(\ell)}_{\mathsf{fmt\text{-}swk}}$ sampled uniformly in R^2_{PQ} (and possibly seeded), and

$$\mathbf{B}^{(\ell)}_{\mathsf{fmt\text{-}swk}} = -\mathbf{a}^{(\ell)}_{\mathsf{fmt\text{-}swk}} \cdot \mathsf{sk}^{(\ell)} + \mathbf{E}^{(\ell)} + P \cdot \begin{pmatrix} \mathsf{sk}_0^{(\ell-1)} \ldots \mathsf{sk}_{2^{\ell-1}-1}^{(\ell-1)} & \\ & \mathsf{sk}_0^{(\ell-1)} \ldots \mathsf{sk}_{2^{\ell-1}-1}^{(\ell-1)} \end{pmatrix} \in R_{PQ}^{2\times 2^\ell} \ ,$$

where $\mathbf{E}^{(\ell)}$ has small-magnitude coefficients.

Now, given as input the ciphertexts $\{(a_i, b_i)\}_{0\le i<k}$, the format conversion proceeds as follows. We define, for all $\ell \le \kappa$ and $0 \le i \le 2^{\kappa-\ell}-1$:

$$\begin{aligned} a_i^{(\ell)} &= \left\lfloor \frac{1}{P}\Big(\langle (a_{2i}^{(\ell-1)}, a_{2i+1}^{(\ell-1)})^T, \mathbf{a}^{(\ell)}_{\mathsf{fmt\text{-}swk}} \rangle \bmod PQ \Big) \right\rceil \ , \\ (b_{i2^\ell}^{(\ell)}, \ldots, b_{i2^{\ell+1}-1}^{(\ell)}) &= (b_{i2^\ell}^{(\ell-1)}, \ldots, b_{i2^{\ell+1}-1}^{(\ell-1)}) \\ &\quad + \left\lfloor \frac{1}{P}\Big((a_{2i}^{(\ell-1)}, a_{2i+1}^{(\ell-1)}) \cdot \mathbf{B}^{(\ell)}_{\mathsf{fmt\text{-}swk}} \bmod PQ \Big) \right\rceil . \end{aligned}$$

Assume that $a_i \cdot \mathsf{sk} + b_i \approx m_i \bmod Q$ for all $0 \le i < n$. It may be checked by induction that we have, for all $\ell \le \kappa$, $0 \le i < 2^{\kappa-\ell}$ and $0 \le j < 2^\ell$:

$$a_i^{(\ell)} \cdot \mathsf{sk}_j^{(\ell)} + b_{i2^\ell+j}^{(\ell)} \approx m_{i2^\ell+j} \bmod Q \ .$$

For $\ell = \kappa$, the latter equation provides the correctness of the format conversion. Letting $a' = a_0^{(\kappa)}$ and $b'_i = b_i^{(\kappa)}$, we have that $a' \cdot \mathsf{sk}'_i + b'_i \approx m_i \bmod Q$, for all $0 \le i < n$.

Compared to the approach of Subsection 3.2, this variant has a format conversion key size that grows only linearly with the number n of ciphertexts. Further, its cost is dominated by $O(n \log n)$ multiplications in $\mathcal{R}_{PQ}$, compared to $O(n^2)$. However, this asymptotic improvement may not show in practice as this method involves more changes of modulus (from Q to PQ and backwards), which is relatively costly.

3.4 Backward format conversion

Suppose some computations have been run on shared-a ciphertexts, initially under keys $(\mathsf{sk}'_i)_{0\le i<n}$ and we want to convert back to ciphertexts for a share secret sk. This may be implemented using the $\mathsf{KeySwitch}$ algorithm; we shall let the corresponding procedure be denoted by $\mathsf{Backward\text{-}FmtSwitch}$. We observe that this introduces a circular security assumption, as the format conversion key from sk to the sk_i's contains RLWE encryptions of $P \cdot \mathsf{sk}$ under the sk'_i's, and the backward format conversion keys contain RLWE encryptions of $P \cdot \mathsf{sk}'_i$'s under sk.

3.5 RLWE ciphertexts with subring a-part

Assume we have a ciphertext $(a, b) \in \mathcal{R}^2_{Q,N}$ for a secret key sk. Using the ring-switching technique described in Section 2.3 allows to turn (a, b) into n ciphertexts in $\mathcal{R}_{Q,N'}$. We can then use the techniques above to have them share their a-parts.

This approach does not require to have multiple ciphertexts at the beginning, but relies on the observation that a large-ring ciphertext may be viewed as several subring ciphertexts and these subring ciphertexts can be transformed into ciphertext with shared-a in a subring when the modulus is sufficiently small.

4 Matrix multiplication algorithms

In this section, we describe new reductions from PC-MM to PP-MM's, for diverse matrix dimensions $d \geq \Omega(N^{1/2+\varepsilon})$. These rely on matrix equations that preserve the formats of the ciphertexts when multiplying by the plaintext matrix. These formats differ from standard RLWE ciphertexts: the MLWE format is interesting for small-dimensional PC-MM, and shared-a RLWE is efficient for large-dimensional PC-MM.

The conversion between RLWE ciphertexts and shared-a RLWE ciphertexts was described in Section 3. Conversions between MLWE and RLWE formats have been described in [2] (see Section 2). All conversions require $\widetilde{O}(dN)$ bit operations, which is asymptotically less significant than the cost for matrix multiplication for $d = \Omega(N^{1/2+\varepsilon})$ (under the assumption that matrix multiplication has a cubic cost).

We focus on coefficients-encoded ciphertexts at modulus $Q_1 = q_0 q_1$. For simplicity, we restrict the discussion to square matrices, although our algorithms extend to rectangular matrices. For the rectangular matrices, the number of columns of the encrypted matrix determines the choice of algorithm to use. In particular, if it is larger (resp. smaller) than N, we can rely on the multi-secret shared-a RLWE (resp. MLWE) format.

4.1 Multi-secret RLWE for large dimensions

For large matrix dimensions, with $d \geq N$, the cost of PP-MM is not negligible. Hence, reducing a single floating-point $d \times d \times d$ PC-MM to two modular $d \times d \times d$ PP-MM may not be satisfactory. Instead, we give a reduction to one $d \times d \times d$ modular PP-MM and one $d \times d \times N$ modular PP-MM. For this purpose, we rely on multi-secret shared-a RLWE. We assume that $d = nN$ for some integer n.

Matrix view of multi-secret RLWE. Consider coefficients-encoded shared-a RLWE ciphertexts $(a, b_j = -a \cdot \mathsf{sk}_j + m_j) \in \mathcal{R}^2_{Q_1,N}$ for $0 \leq j < n$, for messages $m_0, \ldots, m_{n-1}$ and secret keys $\mathsf{sk}_0, \ldots, \mathsf{sk}_{n-1}$. We have the following equation, modulo Q_1:

$$\mathbf{a}^T \cdot \left[\mathtt{Toep}(\mathsf{sk}_0) | \cdots | \mathtt{Toep}(\mathsf{sk}_{n-1})\right] + [\mathbf{b}_0^T | \cdots | \mathbf{b}_{n-1}^T] \;=\; [\mathbf{m}_0^T | \cdots | \mathbf{m}_{n-1}^T] \;,$$

where $\mathbf{a}$, the $\mathbf{b}_i$'s and the $\mathbf{m}_i$'s are N-dimensional vectors corresponding to a, the b_i's and the m_i's, respectively.

Now, to encrypt the $d \times d$ matrix $\mathbf{M}$, we encrypt each row using the above process, with one a-part for the row and n secret keys, which are the same across all rows. More concretely, we consider the ciphertexts $(a_i, b_{i,j} = -a_i \cdot$

$\mathsf{sk}_j + m_{i,[jN,(j+1)N)}) \in \mathcal{R}^2_{Q_1,N}$ for $0 \leq i < d$ and $0 \leq j < n$. Here, the notation $m_{i,[jN,(j+1)N)}$ refers to the ring element whose coefficients are $m_{i,jN}, \ldots,$ $m_{i,(j+1)N-1}$. We call this a $d \times n$ bundle of shared-a RLWE ciphertexts.

Lemma 2. *For any $d \times d/N$ bundle of shared-a RLWE ciphertexts $(a_i, b_{i,j} = -a_i \cdot \mathsf{sk}_j + m_{i,[jN,(j+1)N))}) \in \mathcal{R}^2_{Q_1,N}$ for $0 \leq i < d$ and $0 \leq j < n$, there exist a $d \times N$ matrix $\mathbf{A}$ and a $d \times d$ matrix $\mathbf{B}$ over $\mathbb{Z}_{Q_1}$ such that*

$$\mathbf{A} \cdot \left[\mathtt{Toep}(\mathsf{sk}_0) | \cdots | \mathtt{Toep}(\mathsf{sk}_{d/N-1})\right] + \mathbf{B} \;=\; \mathbf{M} \;. \tag{7}$$

Conversely, if such a matrix equation holds, then there exists a $d \times d/N$ bundle of shared-a RLWE ciphertexts $(a_i, b_{i,j})$ such that $b_{i,j} = m_{i,[jN,(j+1)N)} - a_i \cdot \mathsf{sk}_j \in \mathcal{R}^2_{Q_1,N}$ for $0 \leq i < d$ and $0 \leq j < d/N$.

This lemma is a direct consequence of the matrix form of the polynomial product in $\mathcal{R}_{Q_1,N}$, which we state now. For $u = \sum_{i=0}^{N-1} u_i X^i \in R_{Q_1,N}$, we define $\mathtt{Vec}(u) \in \mathbb{Z}^N_{Q_1}$ to be the N-dimensional row vector $[u_0, \ldots, u_{N-1}]$; $\mathtt{Vec}()$ is obviously a linear isomorphism.

Lemma 3. *Let $u, s, v, w \in R_{Q_1,N}$. The following identities are equivalent:*

$$us + v = w \Leftrightarrow \mathtt{Vec}(u)\mathtt{Toep}(s) + \mathtt{Vec}(v) = \mathtt{Vec}(w).$$

Proof. We rewrite the identity $w = us + v$ under the equivalent form $w = v + \sum_{i=0}^{N-1} u_i \cdot (X^i s)$. By taking $\mathtt{Vec}()$ of both sides, we get

$$\mathtt{Vec}(w) = \mathtt{Vec}(v) + \sum_{i=0}^{N-1} u_i \mathtt{Vec}(X^i s).$$

By definition, $\mathtt{Vec}(X^i s)$ is the i-th row of $\mathtt{Toep}(s)$. The last sum is thus equal to $\mathtt{Vec}(u)\mathtt{Toep}(s)$, as claimed. □

Proof of Lemma 2. We apply Lemma 3 to $u = a_i$, $v = \mathsf{sk}_j$ and get

$$\mathtt{Vec}(a_i) \cdot \mathtt{Toep}(\mathsf{sk}_j) + \mathtt{Vec}(b_{ij}) = \mathtt{Vec}(m_{ij}),$$

for $0 \leq i < d$, $0 \leq j < d/N$. Stacking the a_i vertically, the sk_j horizontally, and the b_{ij} and $m_{i,[jN,(j+1)N)}$'s both ways yields Equation (7).

Conversely, if such a matrix equation holds, we define the ring elements a_i from the i-th row of $\mathbf{A}$ for each $i, 0 \leq i < d$, the secret key sk_j from the j-th Toeplitz matrix for each $j, 0 \leq j < d/N$. Finally, we define the ring elements $b_{i,j}$ (resp. $m_{i,[jN,(j+1)N)}$) from the j-th piece of length N of the i-th row of $\mathbf{B}$ (resp. $\mathbf{M}$). Then, Equation (7) is equivalent to $b_{i,j} = m_{i,[jN,(j+1)N)} - a_i \cdot \mathsf{sk}_j \in \mathcal{R}^2_{Q_1,N}$, which is a $d \times d/N$ bundle of RLWE ciphertexts. □

PC-MM algorithm for large dimensions. We now use Equation (7) to design a PC-MM algorithm. It consists in multiplying both sides of Equation (7) from the left by the plaintext matrix $\mathbf{U}$. To be more precise, suppose we are given

a $d \times k$ bundle of shared-a RLWE ciphertexts encrypting the $d \times d$ matrix $\mathbf{M}$. By left-multiplying by a $d \times d$ matrix $\mathbf{U}$, we obtain the following matrix equation:

$$(\mathbf{UA}) \cdot \left[\mathtt{Toep}(\mathsf{sk}_0) | \cdots | \mathtt{Toep}(\mathsf{sk}_{k-1})\right] + (\mathbf{UB}) \;=\; (\mathbf{UM}).$$

By Lemma 2, the result corresponds to a $d \times k$ bundle of shared-a RLWE ciphertexts encrypting the $d \times d$ matrix $\mathbf{UM}$. The cost consists in a $d \times d \times N$ PP-MM of $\mathbf{U}$ and $\mathbf{A}$ and a $d \times d \times d$ PP-MM of $\mathbf{U}$ with $\mathbf{B}$.

For PC-MM with inputs and outputs of ordinary RLWE formats, we first (forward) convert the format so that the a-part involved in each given row matches, for secrets $\mathsf{sk}_0, \ldots, \mathsf{sk}_{k-1}$. For this purpose, we use either conversion algorithm described in Section 3. Then, we compute PC-MM as described above. Finally, we (backward) convert the format, obtaining RLWE ciphertexts with a shared secret sk, encrypting the matrix $\mathbf{U}\dot{\mathbf{M}}$. To accelerate backward conversion, we rescale the ciphertexts after PC-MM, even though it is not necessary for a standalone PC-MM. Algorithm 1 describes the entire process.

Algorithm 1 PC-MM for dimension above the RLWE ring degree N

Input: A matrix $\mathbf{U} \in \mathbb{R}^{d \times d}$, with $d = kN$ for some integer $k \geq 1$,
coefficients-encoded RLWE ciphertexts $(\mathsf{ct}_i)_{0 \leq i < d^2/N}$ in $\mathcal{R}_{Q_1,N}$,
each of which encrypts a segment of a row of a matrix $\mathbf{M} \in \mathbb{R}^{d \times d}$.
Input: Switching keys $\mathsf{fmt\text{-}swk}_{Q_1,P,\mathsf{sk} \to \{\mathsf{sk}_i\}_{0 \leq i < n}}$ and $\mathsf{swk}_{Q_0,P,\mathsf{sk}_i \to \mathsf{sk}}$ for $0 \leq i < n$.
Output: Coefficients-encoded RLWE ciphertexts in $\mathcal{R}_{Q_0,N}$,
each of which encrypts a segment of a row of the matrix $\mathbf{UM} \in \mathbb{R}^{d \times d}$.

1: **for** $i \leftarrow 0$ **to** d **do**
2: $\quad (a_i, b_{i,j}) \leftarrow \mathsf{FmtSwitch}(\mathsf{ct}_{ni}, \cdots, \mathsf{ct}_{n(i+1)-1}; \mathsf{fmt\text{-}swk}_{Q_1,P,\mathsf{sk} \to \{\mathsf{sk}_i\}_{0 \leq i < n}})$
3: **end for**
4: $(\mathbf{A}, \mathbf{B}) \leftarrow ((a_{i,j})_{0 \leq i < d, 0 \leq j < N}, (b_{i,j})_{0 \leq i < d, 0 \leq j < d})$
5: $(\mathbf{A}', \mathbf{B}') \leftarrow (\mathbf{U} \cdot \mathbf{A}, \mathbf{U} \cdot \mathbf{B})$
6: **for** $i \leftarrow 0$ **to** d **and** $j \leftarrow 0$ **to** d/N **do**
7: $\quad \mathsf{ct}'_{i,j} \leftarrow (\sum_k a'_{i,k} X^k, \sum_k b'_{i,jN+k} X^k) \in R^2_{Q_1,N}$
8: $\quad \mathsf{ct}'_{i,j} \leftarrow \mathsf{Rescale}(\mathsf{ct}'_{i,j}; Q_1, Q_0)$
9: $\quad \mathsf{ct}'_{i,j} \leftarrow \mathsf{KeySwitch}(\mathsf{ct}'_{i,j}; \mathsf{swk}_{Q_0,P,\mathsf{sk}_i \to \mathsf{sk}})$
10: **end for**
11: **return** all $(\mathsf{ct}')_{i,j}$'s

Besides the $d \times d \times N$ and $d \times d \times d$ PP-MM's, the algorithm involves $O(d)$ format conversion key switchings with $n = d/N$ secrets (Step 2), $O(d^2/N)$ ciphertext rescalings (Step 8) and $O(d^2/N)$ key switchings (Step 9). The non-PP-MM costs hence grow as $\widetilde{O}(d^2 \log Q_1)$ bit operations. This is negligible compared to the cost $\widetilde{O}(d^\omega \log Q_1)$ of matrix multiplication over $\mathbb{Z}_{Q_1}$. Overall, for d large compared to N, the cost is almost only that of one PP-MM in the same dimensions as the PC-MM under scope.

We note that in some scenarios, it may be preferable not to perform the backward format conversion (Step 9) and keep ciphertexts that share their a-parts.

On the other side, there exist scenarios in which we already have ciphertexts sharing a-parts, which enables us to skip forward format conversion (Step 2).

4.2 MLWE for small dimensions

For matrices of dimension d below the ring degree N, the matrix multiplication requires a higher level of granularity than that of the RLWE ring. If fully using the coefficients of the RLWE ciphertexts, then several columns, rows or diagonals of the encrypted matrix are stored in a single ciphertext. This makes homomorphic computations more cumbersome, possibly requiring permutations and maskings to extract relevant data from the ciphertexts. This requires key switching within the matrix multiplication algorithm, which incurs a computational overhead, as it prevents from reducing directly to PP-MM and benefit from highly efficient PP-MM software.

Matrix view of MLWE. Assume we are given a coefficients-encoded MLWE ciphertext $((a_j)_{0\leq j<k}, b = -\sum_j a_j \mathsf{sk}_j + m)$, where $a_j \in \mathcal{R}_{Q_1,N}$ for all j. Then we have the following equation modulo Q_1:

$$\mathbf{a}^T \cdot \begin{bmatrix} \frac{\mathtt{Toep}(\mathsf{sk}_0)}{\vdots} \\ \hline \mathtt{Toep}(\mathsf{sk}_{k-1}) \end{bmatrix} + \mathbf{b}^T \;=\; \mathbf{m}^T \;, \tag{8}$$

where $\mathbf{b}$ and $\mathbf{m}$ are the vectors corresponding to the (degree-N) ring element b and m, and $\mathbf{a}$ is the concatenation of the vectors corresponding to the a_j's. When we have multiple MLWE ciphertexts with a shared secret key $\mathbf{sk} = (\mathsf{sk}_j)_{0\leq j<k}$, Equation (8) becomes a matrix equation.

Lemma 4. *Let* $\mathsf{ct}_i = ((a_{i,j})_{0\leq j<k}, b_i = -\sum_j a_{i,j}\mathsf{sk}_j + m_{i,[0,d)})$ *be* n *coefficients-encoded* $\mathsf{MLWE}^k_{Q_1,d}$ *for the same secret key* $\mathbf{sk} = (\mathsf{sk}_j)_{0\leq j<k}$. *Then there exist an* $n \times kd$ *matrix* $\mathbf{A}$ *and an* $n \times d$ *matrix* $\mathbf{B}$ *over* $\mathbb{Z}_{Q_1}$ *such that*

$$\mathbf{A} \cdot \begin{bmatrix} \frac{\mathtt{Toep}(\mathsf{sk}_0)}{\vdots} \\ \hline \mathtt{Toep}(\mathsf{sk}_{k-1}) \end{bmatrix} + \mathbf{B} \;=\; \mathbf{M} \;. \tag{9}$$

Conversely, if such a matrix equation holds, then there exist n *coefficients-encoded* $\mathsf{MLWE}^k_{Q_1,d}$ *ciphertexts* $\mathsf{ct}_i = ((a_{i,j})_{0\leq j<k}, b_i)$ *such that* $b_i = -\sum_j a_{i,j}\mathsf{sk}_j + m_{i,[0,d)}$ *for all* i.

The proof follows from applying Lemma 3 to $u = a_{ij}$, $s = \mathsf{sk}_j$, $v = b_i$, $w = m_{i,[0,d)}$ and suitably stacking the matrix identity obtained for $0 \leq i < n$, $0 \leq j < k$.

PC-MM algorithm for small dimensions. Similarly to the large-dimensional case, Equation (9) provides a reduction from PC-MM (with MLWE ciphertexts) to two modular PP-MM. Assume we are given d MLWE ciphertexts of degree d

and rank k that encrypt a matrix $\mathbf{M}$, row by row. Here k is set so that $\mathsf{MLWE}^k_{Q_1,d}$ is hard. To multiply $\mathbf{M}$ by the $d \times d$ plaintext matrix $\mathbf{U}$, we multiply both sides of Equation (9) on the left by $\mathbf{U}$. We obtain the following matrix equation:

$$(\mathbf{UA}) \cdot \left[\begin{array}{c} \mathtt{Toep}(\mathsf{sk}_0) \\ \hline \vdots \\ \hline \mathtt{Toep}(\mathsf{sk}_{k-1}) \end{array}\right] + (\mathbf{UB}) \;=\; (\mathbf{UM}) \;.$$

By Lemma 4, the result corresponds to d MLWE ciphertexts of rank k over $\mathcal{R}_{Q_1,d}$. The cost is that of a $d \times d \times dk$ PP-MM and a $d \times d \times d$ PP-MM.

For PC-MM with ordinary degree N RLWE inputs and outputs, we use the conversions between degree N RLWE and $\mathsf{MLWE}^k_{Q_1,d}$ (see Section 2), along with the above reduction to PP-MM. Algorithm 2 describes the entire process. It requires that $d \geq N^{1/2}$.

Besides the $d \times d \times N$ and $d \times d \times d$ PP-MM's, the algorithm involves $O(d^2/N)$ conversions from RLWE-format ciphertexts in degree $N = kd$ to MLWE-format ciphertexts with degree d and rank k (Step 2), as well as $O(d^2/N)$ conversions from MLWE-format ciphertexts with the same parameters to RLWE-format ciphertext (Step 14). The non-PP-MM costs hence grow as $\widetilde{O}(d^2 \log Q_1)$ bit operations. This is negligible compared to the cost $\widetilde{O}((1 + N/d) \cdot d^\omega \log Q_1)$ of matrix multiplication.

Note that MLWE-format ciphertexts can be used in place of RLWE-format ciphertexts when transferring data from client to server, with enhanced granularity (see [2]). In such a scenario, the first conversion, from RLWE to MLWE, may hence be skipped.

4.3 Faster PC-MM with precomputation

We now design a reduction from PC-MM to one PP-MM, with precomputation. Importantly, we can leverage the technique from Section 5 so that the PP-MM is indeed a floating-point PP-MM rather than a modular PP-MM. This then gives an optimal reduction from PC-MM to a single floating-point PP-MM.

Another matrix view of multi-secret RLWE. Consider coefficients-encoded shared-a RLWE ciphertexts $(a, b_j = -a \cdot \mathsf{sk}_j + m_j) \in \mathcal{R}^2_{Q_1,N}$ for $0 \leq j < n$, for messages $m_0, \ldots, m_{n-1}$ and secret keys $\mathsf{sk}_0, \ldots, \mathsf{sk}_{n-1}$. We have the following equation, modulo Q_1:

$$\left[\begin{array}{c} \mathsf{sk}_0^T \\ \hline \vdots \\ \hline \mathsf{sk}_{n-1}^T \end{array}\right] \cdot \mathtt{Toep}(a) + \left[\begin{array}{c} \mathbf{b}_0^T \\ \hline \vdots \\ \hline \mathbf{b}_{n-1}^T \end{array}\right] = \left[\begin{array}{c} \mathbf{m}_0^T \\ \hline \vdots \\ \hline \mathbf{m}_{n-1}^T \end{array}\right] \;,$$

where we identified sk_i, b_i and m_i with the column vector corresponding to its coefficients. To encrypt $d \times d$ matrices with $d \geq N$, we proceed as in Section 4.1. We obtain the following lemma:

Algorithm 2 PC-MM for dimension below the RLWE ring degree N

Input: A matrix $\mathbf{U} \in \mathbb{R}^{d \times d}$, with $N = kd$ for some integer $k \geq 1$,
coefficients-encoded RLWE ciphertexts $(\mathsf{ct}_i)_{0 \leq i < d^2/N}$ in $\mathcal{R}_{Q_1,N}$,
each of which encrypts a strip of rows of a matrix $\mathbf{M} \in \mathbb{R}^{d \times d}$.
Input: Switching keys for $\mathsf{ModPack}$
Output: Coefficients-encoded RLWE ciphertexts in $\mathcal{R}_{Q_0,N}$,
each of which encrypts a strip of rows of the matrix $\mathbf{UM} \in \mathbb{R}^{d \times d}$.

for $i \leftarrow 0$ **to** d^2/N **do**
  $((\mathbf{a}_{di+j}, b_{di+j}))_{0 \leq j < k} \leftarrow \mathsf{ModDecomp}(\mathsf{ct}_i)$
end for
for $i \leftarrow 0$ **to** d **do**
  Set the i-th row of $\mathbf{A}$ as the concatenation of coefficients of $\mathbf{a}_i$
  Set the i-th row of $\mathbf{B}$ as the coefficients of b_i
end for
$(\mathbf{A}', \mathbf{B}') \leftarrow (\mathbf{UA}, \mathbf{UB})$
for $i \leftarrow 0$ **to** d **do**
  Set the concatenation of coefficients of $\mathbf{a}'_i$ as the i-th row of $\mathbf{A}'$
  Set the coefficients of b_i as the i-th row of $\mathbf{B}'$
  Rescale $((\mathbf{a}'_i, b_i); Q_1, Q_0)$
end for
for $i \leftarrow 0$ **to** d^2/N **do**
  $\mathsf{ct}'_i \leftarrow \mathsf{ModPack}((a'_{di+j}, b'_{di+j})_{0 \leq j < k})$
end for
return all ct'_i's

Lemma 5. *For any given $d/N \times d$ bundle of shared-a RLWE ciphertexts $(a_i, b_{i,j} = -a_i \cdot \mathsf{sk}_j + m_{i,[jN,(j+1)N)}) \in \mathcal{R}^2_{Q_1,N}$ for $0 \leq i < d/N$ and $0 \leq j < d$, there exists a $d \times N$ matrix $\mathbf{S}$ and a $d \times d$ matrix $\mathbf{B}$ over $\mathbb{Z}_{Q_1}$ such that*

$$\mathbf{S} \cdot \left[\mathtt{Toep}(a_0) | \cdots | \mathtt{Toep}(a_{d/N-1}) \right] + \mathbf{B} \;=\; \mathbf{M} \;. \tag{10}$$

Conversely, if such a matrix equation holds, then there exists a $d/N \times d$ bundle of shared-a RLWE ciphertexts $(a_i, b_{i,j})$ such that $b_{i,j} = -a_i \cdot \mathsf{sk}_j + m_{[i,jN..(j+1)N)} \in \mathcal{R}^2_{Q_1,N}$ for $0 \leq i < d/N$ and $0 \leq j < d$.

The proof follows from applying Lemma 3 to $u = \mathsf{sk}_j$, $s = a_i$, $v = b_{i,j}$, $w = m_{i,[jN,(j+1)N)}$ and suitably stacking the matrix identity obtained for $0 \leq i < d/N$, $0 \leq j < d$.

Precomputation PC-MM algorithm for large dimensions. By multiplying both sides of Equation (10) on the left by $\mathbf{U}$, we obtain

$$(\mathbf{US}) \cdot \left[\mathtt{Toep}(a_0) | \cdots | \mathtt{Toep}(a_{d/N-1}) \right] + (\mathbf{UB}) \;=\; (\mathbf{UM}) \;. \tag{11}$$

Lemma 5 implies that this is equivalent to shared-a RLWE ciphertexts, with secret keys corresponding to each row of $\mathbf{US}$.

Observe that **US** is independent of the ciphertext, and depends only on the secret keys. Theorefore, if **U** is known in advance, we can *precompute* the key-switching key from each row of **US** to a common secret key, in an off-line phase. Then, in the online phase, a single PP-MM **UB** completes the PC-MM with shared-a RLWE ciphertexts.

Assuming the inputs is given by RLWE ciphertexts, we first (forward) convert the ciphertexts to shared-a ciphertexts, where the sharing is across a matrix row, and for secrets $\mathsf{sk}_0, \cdots \mathsf{sk}_{n-1}$. We then compute a single PP-MM for **UB** and do not perform any on-line computation for the a-part. Finally, we (backward) convert to RLWE ciphertexts for a shared secret sk, using pre-computed switching keys from **US**. This is formalized in Algorithm 3.

When **U** is not known at setup, it is also possible to generate the switching keys from **US** to sk after **U** is known. For given switching keys for backward format conversion which correspond to the rows of **S**, we can perform PP-MM to the switching keys, obtaining the desired key from **US** to sk.

In this precomputation-based algorithm (and the one described below), we have secret keys for **US**, which do not have small magnitude coefficients. To keep the rescaling error small, we switch the secret to a small secret (e.g., sparse ternary secret) before any rescaling.

Another matrix view of multi-secret MLWE. For matrices of size $d \times d$ with $d < N$, we use shared-a MLWE formats to reduce PC-MM to a single PP-MM, with precomputation. Consider coefficients-encoded shared-a MLWE ciphertexts $(\mathbf{a}, b_j = -\langle \mathbf{a}, \mathbf{sk}_j \rangle + m_j) \in \mathcal{R}^{k+1}_{Q_1,N}$ for $0 \le j < n$ for some n, for messages m_j's and secret keys $\mathbf{sk}_j$'s. We have the following equation modulo Q_1:

$$\begin{bmatrix} \mathbf{sk}_0^T \\ \hline \vdots \\ \hline \mathbf{sk}_{n-1}^T \end{bmatrix} \cdot \begin{bmatrix} \mathtt{Toep}(a_0) \\ \hline \vdots \\ \hline \mathtt{Toep}(a_{k-1}) \end{bmatrix} + \begin{bmatrix} \mathbf{b}_0^T \\ \hline \vdots \\ \hline \mathbf{b}_{n-1}^T \end{bmatrix} = \begin{bmatrix} \mathbf{m}_0^T \\ \hline \vdots \\ \hline \mathbf{m}_{n-1}^T \end{bmatrix},$$

where the $\mathbf{b}_j$'s and the $\mathbf{m}_j$'s are N-dimensional vectors corresponding to the b_j's, the m_j's, respectively; the rows of the left matrix are kN-dimensional vectors corresponding to $\mathbf{sk}_i \in \mathcal{R}^k_{Q_1,N}$ for all $0 \le i < n$, and a_j is the j-th ring element of $\mathbf{a}$'s for all $0 \le j < k$.

To encrypt a $d \times d$ matrix $\mathbf{M}$ with d dividing N, we consider shared-a MLWE ciphertext $(\mathbf{a}, b_j = -\langle \mathbf{a}, \mathbf{sk}_j \rangle + m_j) \in \mathcal{R}^{k+1}_{Q_1,d}$ for $0 \le j < d$, where m_j corresponds to the j-th row of $\mathbf{M}$. Note that the security of MLWE is determined by dk.

Lemma 6. *For any d shared-a MLWE ciphertexts $(\mathbf{a}, b_j = -\langle \mathbf{a}, \mathsf{sk}_j \rangle + m_j) \in \mathcal{R}^{k+1}_{Q_1,d}$ for $0 \le j < d$, there exist a $d \times N$ matrix $\mathbf{S}$ and a $d \times d$ matrix $\mathbf{B}$ over $\mathbb{Z}_{Q_1}$ such that*

$$\mathbf{S} \cdot \begin{bmatrix} \mathtt{Toep}(a_0) \\ \hline \vdots \\ \hline \mathtt{Toep}(a_{N/d-1}) \end{bmatrix} + \mathbf{B} \;=\; \mathbf{M} \;. \tag{12}$$

Conversely, if such a matrix equation holds, then there exist d shared-a MLWE ciphertexts (a, b_j) such that $b_j = -\langle \mathbf{a}, \mathbf{sk}_j \rangle + m_j \in \mathcal{R}^{k+1}_{Q_1,d}$ for $0 \le j < d$.

This again follows from Lemma 3 for $u = \mathsf{sk}_{ij}$, $s = a_i$, $v = b_j$, $w = m_j$, $0 \le i < N/d$, $0 \le j < d$, and stacking the matrix identity thus obtained.

Precomputation PC-MM algorithm for small dimensions. Similarly to the large-dimensional case, we can reduce PC-MM to a PP-MM of **UB**, by allowing the precomputation of **US** on the key material. Concretely, by multiplying both sides of Equation (12) on the left, we have:

$$(\mathbf{US}) \cdot \begin{bmatrix} \mathtt{Toep}(a_0) \\ \hline \vdots \\ \hline \mathtt{Toep}(a_{k-1}) \end{bmatrix} + (\mathbf{UB}) \;=\; (\mathbf{UM}) \;.$$

During the online phase, we evaluate the PP-MM corresponding to the b-part, and do not change the a-part. Switching keys from **US** to a common RLWE secret can be precomputed.

To apply this to ordinary RLWE-format ciphertexts, we first convert RLWE-format ciphertexts to shared-a MLWE ciphertexts. This is achieved by converting from RLWE ciphertexts to shared-a RLWE ciphertexts and then from RLWE ciphertexts to MLWE ciphertexts using the ShModDecomp procedure.

The opposite direction also can be achieved in two steps: (a) go from shared-a MLWE ciphertexts to shared-secret MLWE ciphertexts by using ModKeySwitch (see Section 2.3), and (b) run ModPack to obtain RLWE ciphertexts. The whole process is described in Algorithm 4, which requires that $d \ge N^{1/2}$.

For the backward conversion, it is even possible to convert shared-a MLWE ciphertexts into ordinary RLWE in one step, if we have a key-switching key from the column of **US** to a common RLWE secret.

5 Optimizing the *b*-part matrix multiplication

The algorithms from Section 4 rely on PP-MM's modulo an integer which is small but can still be larger than a machine word. Note that modular PP-MM is not as fast as floating-point PP-MM, which benefits from dedicated libraries such as OpenBLAS [28]. For our implementation, we reduce PP-MM modulo an integer to several floating-point PP-MM's (see Section 7.1).

To reduce the cost of the PP-MM's, we explain below how to implement the PP-MM corresponding to the b-parts of the ciphertexts, i.e., the evaluation of **UB**, with a floating-point PP-MM rather than a modular one. In the precomputation setting (see Subsection 4.3), this gives a reduction from one floating-point PC-MM to one floating-point PP-MM.

Note that the b-part of RLWE/MLWE ciphertexts contains the information of the plaintext message in specific positions of bits. In particular, a CKKS ciphertext contains the plaintext message only in the most significant bits of the b-part, while the least significant bits consist of numerical errors and RLWE or MLWE errors that are irrelevant. Therefore, for the b-part $\mathbf{U} \cdot \mathbf{B} \bmod Q_1$, we can multiply the most significant bits of **U** with the most significant bits of **B** and keep the most significant bits of $\mathbf{U} \cdot \mathbf{B}$ (without reduction modulo Q_1).

This can be achieved with double-precision floating-point PP-MM, unless the required target precision is too large. The required precision is driven by the target precision and the overflow modulo Q_1. Chopping off the bits of the b-part is a classical technique in lattice-based cryptography, used for example in Kyber [3] and dating back to as far as [30]. Here we extend this technique to the context of PC-MM.

We first explain how to use floating-point arithmetic to multiply a scalar of the b-part of a single ciphertext. Suppose we are given $b \in \mathcal{R}_{Q_1,N}$, where $Q_1 = q_0 q_1$. Note that the prime q_1 is used solely for PC-MM, while q_0 is possibly related to other homomorphic computations beyond scalar multiplications (e.g., bootstrapping). From this, we note that q_1 can be much smaller than q_0 (e.g., we take $q_1 \approx 2^{18}$ and $q_0 \approx 2^{58}$ in our experiments). To multiply b by a scalar $u \in \mathbb{R}$:

- we first extract the first 53 most significant bits of b, to obtain a floating-point number $\widetilde{b}$;
- we then multiply it with the encoding $\mathsf{Ecd}(u) \in \mathbb{Z}$ of u with scaling factor $\Delta_1 = q_1$ (i.e., $\mathsf{Ecd}(u)$ has magnitude $\approx q_1$); let y denote the result;
- finally, the modular reduction by q_0 completes the entire multiplication; it is performed by dividing y by $q_0 q_1$, taking the fractional part, and multiplying by q_0.

Note that $\widetilde{b}$ contains the first 53-bits of b and has magnitude $\approx Q_1 = q_0 q_1$. The quantity y is about q_1 times larger, and corresponds to an approximation of $b \cdot \mathsf{Ecd}(u)$ over the reals, to ≈ 53 bits of precision. Dividing by $q_0 q_1$ maintains this precision, but the output is now of magnitude $\approx q_1$. Taking the fractional part leads to a loss of $\approx \log q_1$ bits of precision. This precision is maintained while re-multiplying by q_0. Overall, this gives an approximation to $b' = (\mathsf{Ecd}(u) \cdot b \bmod q_0 q_1)/q_1$ to a precision of $\approx 53 - \log q_1$ bits. Figure 1 visualizes this process.

Now, assume that the scaling factor of m in b was Δ. Then so is the scaling factor of $m \cdot \mathsf{Ecd}(u)$ in b' (as we set $\Delta_1 = q_1$). This means that in our approximation to b', the quantity $m \cdot \mathsf{Ecd}(u)$ is correct to precision $\approx 53 - \log q_1 - \log(q_0/\Delta)$. As $\mathsf{Ecd}(u)$ approximates $\Delta_1 \cdot u$ to $\approx \log q_1$ bits, the plaintext underlying b' approximates $m \cdot u$ to a precision of

$$\approx \quad \min\left(53 - \log q_1 - \log\left(\frac{q_0}{\Delta}\right),\ \log q_1\right) \quad \text{bits} .$$

To compute $\mathbf{UB} \bmod Q_1$, we adopt the same strategy. More precisely, for $d \times d$ matrices, we first extract the first 53 most significant bits of each entry of $\mathbf{B}$, then we perform a floating-point PP-MM with the matrix $\mathbf{U}$ that encodes a real-valued matrix with integers and a scaling factor of $\Delta_1 = q_1$, and finally, we reduce each entry modulo q_0 (by dividing by $q_0 q_1$, taking the fractional part and multiplying by q_0). This requires d^2 modular reductions, which is not significant compared to matrix multiplication in dimension d. Heuristically, we expect a growth of the coefficients by a factor $\sqrt{d}$, which leads to an output precision of

$$\approx \quad \min\left(53 - \log q_1 - \log\left(\frac{q_0}{\Delta}\right) - \frac{1}{2}\log d\ ,\ \log q_1\right) \quad \text{bits} .$$

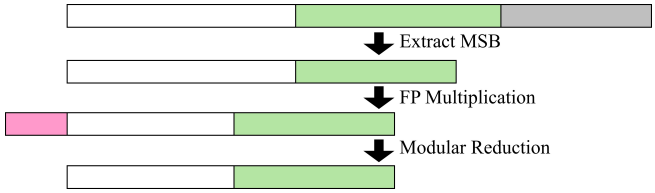

Fig. 1. Illustration of floating-point PP-MM of the b-part. The green areas correspond to the data underlying the ciphertext. The pink area is the $\mod q_0q_1$ overflow due to multiplication by $\mathsf{Ecd}(u)$.

Depending on the matrices to be multiplied, the precision loss could be higher. In particular, in the worst case, the $(\log d)/2$ term should be replaced by $\log d$.

6 Bootstrapping improvements

In this section, we show how the shared-a representation of multiple ciphertexts can be leveraged to accelerate bootstrapping in a situation where several ciphertexts must be bootstrapped at once. This is a common situation in PC-MM, as storing the encrypted input and output matrices often requires several ciphertexts. First, we describe how RLWE key-switching, and thus rotations and conjugations, can be computed while preserving the shared-a representation. We then explain how to fuse bootstrapping with PC-MM, hence providing a form of programmable bootstrapping for matrix multiplication.

6.1 Shared-a homomorphic operations

We start by describing versions of the homomorphic operations Add, PCMult, Rot, Conj which benefit from the shared-a format and preserve it.

Shared-a Add and PCMult. We define the following algorithms.

- Sh-Add. Given as input two n-tuples of shared-a ciphertexts $((a, b_i))_{0\leq i<n}$ and $((a', b'_i))_{0\leq i<n}$ both for secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$, Sh-Add computes and returns $\mathsf{ct}_+ = ((a + a', b_i + b'_i))_{0\leq i<n}$.
- Sh-PCMult. Given as input a n-tuple of shared-a ciphertexts $((a, b_i))_{0\leq i<n}$ and $m \in \mathcal{R}$, Sh-PCMult computes and returns $\mathsf{ct}_\times = ((ma, mb_i))_{0\leq i<n}$.

Note that the outputs of Sh-Add and Sh-PCMult remain in shared-a format.

Further, we note that the Rescale algorithm is compatible with the shared a-part format. Concretely, when applying Rescale to two RLWE ciphertexts (a, b_1) and (a, b_2), the output ciphertexts (a', b'_1) and (a', b'_2) have the same a-part.

In the precomputation algorithms from Section 4, we may end up with shared-a ciphertexts for larger secret keys (as the secret key matrix is multiplied by the plaintext matrix $\mathbf{U}$). This can lead to larger rounding errors when

applying Rescale. To circumvent the difficulty, one may use a shared-a preserving version of the KeySwitch algorithm to switch to smaller-magnitude secret keys.

Shared-a KeySwitch. We start by describing the shared-a switching key format.

- Sh-SWKGen. Given two integers P, Q and two tuples of secret keys $\mathbf{sk} = (\mathsf{sk}_i)_{0\le i<n}$, $\mathbf{sk'} = (\mathsf{sk}'_i)_{0\le i<n} \in \mathcal{R}_N^n$, Sh-SWKGen returns a switching key $\mathbf{Sh\text{-}swk}_{Q,P,\mathbf{sk}\to\mathbf{sk}'} \in \mathcal{R}_{PQ,N} \times \mathcal{R}_{PQ,N}^n$ from shared-a key $\mathbf{sk}$ to shared-a key $\mathbf{sk'}$ for a shared-a ciphertext modulo Q with auxiliary integer P. This key is of the form:

$$\mathbf{Sh\text{-}swk}_{Q,P,\mathbf{sk}\to\mathbf{sk}'} = (a_{\mathbf{Sh\text{-}swk}}, (b_{\mathbf{Sh\text{-}swk},i})_i) \ \in \ \mathcal{R}_{PQ,N} \times \mathcal{R}_{PQ,N}^n \ ,$$

  with $b_{\mathbf{Sh\text{-}swk},i} = -a_{\mathbf{Sh\text{-}swk}} \cdot \mathsf{sk}'_i + e_i + P \cdot \mathsf{sk}_i$, where $e_i \in R_N$ has small-magnitude coefficients, for $0 \le i < n$.

This shared-a switching key can also be viewed as a n-tuple of switching keys with shared a-part. We now describe shared-a key switching.

- Sh-KeySwitch. Given a shared-a switching key $\mathbf{Sh\text{-}swk} = \mathbf{Sh\text{-}swk}_{Q,P,\mathbf{sk}\to\mathbf{sk}'}$, and a n-uple of ciphertexts $(\mathbf{ct}_i)_{0\le i<n}$ in shared-a format, where $\mathbf{ct}_i = ((a, b_i))_{0\le i<n}$ is an encryption of m_i under the secret key $\mathbf{sk}_i$, Sh-KeySwitch returns the shared-a tuple of ciphertexts

$$((a', b'_i))_{0\le i<n} = (\mathsf{KeySwitch}(\mathbf{Sh\text{-}swk}_i, (a, b_i))_{0\le i<n}.$$

It may be checked that the output of Sh-KeySwitch is in shared-a format, and that (a', b'_i) decrypts to m_i under the secret key sk'_i, for all $0 \le i < n$.

The shared-a key-switching procedure also allows to define rotations and conjugation procedures that preserve the shared-a format.

Cost of shared-a operations. We readily observe that the costs of Sh-Add and Sh-PCMult are almost half of those of regular Add and PCMult, when n is large. We turn to the cost of Sh-KeySwitch. From a computational point of view, key-switching can be decomposed into three steps: ModUp (raising from modulo Q to PQ), MultSwk (multiplying), ModDown (lowering the modulus to Q). The first one operates only on the a-part, and can thus be fully batched in shared-a representation, while the last two operate on both parts of the ciphertext. Table 2 compares the number of ring element operations in ModUp, and similarly for MultSwk and ModDown for Sh-KeySwitch over a n-tuple of ciphertexts, to the KeySwitch counterpart. Based on the table, we see that the shared-a key-switching can be expected to be at least twice as fast as the usual key-switching operation, when n is large enough.

It should however be pointed out that, as we use the shared-a format for the switching key, the key size grows linearly with n. To be more specific, the switching key size for rank n shared-a format is $(n+1)/2$ times larger than that of the single secret format. When working at high levels, this implies a large memory footprint.

	ModUp	MultSwk	ModDown
$n\times$ KeySwitch	n	$2n$	$2n$
Sh-KeySwitch	1	$n+1$	$n+1$

Table 2. Numbers of ring operations, for naive key-switching and shared-a keyswitching over n ciphertexts.

6.2 Batch-bootstrapping

We now use the shared-a primitives to improve the efficiency of the S2C and C2S bootstrapping steps. S2C and C2S both correspond to a sequence of products of a plaintext matrix $\mathbf{M}$ with a ciphertext ct, the slots of which are seen as an $N/2$-dimensional vector. The matrix-vector multiplications are followed by a conjugation step and additions. The matrix-vector products are evaluated, using the diagonal method [21], as a sum:

$$\sum_j \mathrm{Diag}_j(\mathbf{M}) \odot \mathsf{Rot}_j(\mathsf{ct}) \ , \tag{13}$$

where $\mathrm{Diag}_j(\mathbf{M}) = (m_{0,j}, \ldots, m_{N/2-j-1,N/2-1}, m_{N/2-j,0}, \ldots, m_{N/2-1,j-1})$ is the j-th diagonal of $\mathbf{M}$.

All the operations used in Equation (13) have a shared-a counterpart, which preserves the shared-a structure. This also holds for the Baby-Step-Giant-Step version of Equation (13). One can thus replace C2S and S2C by shared-a counterparts, which we will call Sh-C2S and Sh-S2C.

It remains to schedule the format switching from ordinary representation to shared-a. For this, we note that the only step of the bootstrapping process which does not have a shared-a preserving version is EvalMod, which consists in the evaluation of a large-degree polynomial, requiring ciphertext-ciphertext multiplications. In the case of S2C-first bootstrap, we thus place the forward format conversion before S2C, and the backward format conversion before EvalMod:

$$\begin{aligned}[\mathsf{FmtSwitch}] \to [\mathsf{Sh\text{-}S2C}] \to [\mathsf{ModRaise}] \to [\mathsf{Sh\text{-}C2S}] \\ \to [\mathsf{Backward\text{-}FmtSwitch}] \to [\mathsf{EvalMod}] \ .\end{aligned}$$

In the case of C2S-first bootstrapping, forward and backward format conversions are inserted at the beginning and end of C2S and S2C, respectively:

$$\begin{aligned}[\mathsf{FmtSwitch}] \to [\mathsf{ModRaise}] \to [\mathsf{Sh\text{-}C2S}] \to [\mathsf{Backward\text{-}FmtSwitch}] \\ \to [\mathsf{EvalMod}] \to [\mathsf{FmtSwitch}] \to [\mathsf{Sh\text{-}S2C}] \to [\mathsf{Backward\text{-}FmtSwitch}] \ .\end{aligned}$$

6.3 Matrix multiplication and bootstrapping

Designing efficient algorithms and implementations in CKKS (the situation is the same in the BFV/BGV context) is a more demanding process than designing "usual" algorithms. Indeed, computations in CKKS have, roughly, a cost that

is linear in $\ell + 1$, where ℓ is the current level of modulus. The algorithm design should always include a scheduling task targetting the placement of computationally intensive steps, such as matrix multiplication, at the lowest available level. This implies that such steps are typically followed by bootstrapping.

We can go further and study the interaction of our PC-MM algorithms with the bootstrapping process. The fact that our algorithms use the *coefficient encoding* points strongly in this direction. Indeed, a ciphertext has a coefficient-encoded message at the lowest level during $\mathsf{S2C}$-first bootstrapping, which most of the current CKKS implementations follow. Putting the PC-MM step at this stage proves to be an optimal solution to the issue of performing PC-MM at a low level.

We prove below that Algorithm 1 ($d \geq N$) also works if the input is slot-encoded. This allows, in this case, other options for the PC-MM computation. While we still believe that fusing linear algebra and bootstrapping remains preferable, we shall also discuss, for the sake of completeness, the other available options.

Since PC-MM (when the matrix dimensions are not small) usually encounters lots of ciphertexts for (simultaneous) bootstrapping, we use the batch bootstrapping techniques of Section 6.2 while fusing PC-MM and bootstrapping. Putting all ingredients together with ring switching, we propose a fast PC-MM algorithm fused with batch bootstrapping, which we call MaMBo (for Matrix Multiplication Bootstrapping).

Compatibility with slots-encoded ciphertexts for $d \geq N$. An important question regarding fusing with other FHE operations is whether the PC-MM algorithms are compatible with slot-encoded ciphertexts. For ease of discussion, let us focus on the simpler case of PC-MM with $d = N$ and without precomputation (i.e., the algorithm from [26]). Recall the following equation from the introduction:

$$\mathbf{A} \cdot \mathtt{Toep}(\mathsf{sk}) + \mathbf{B} = \mathbf{M} \ ,$$

where the rows of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{M}$ correspond to the coefficients of a, b and m parts of the input ciphertexts. We now observe that the coefficients-encoded matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$ is related to the slots-encoded matrix $\mathbf{M}' \in \mathbb{C}^{N \times N/2}$ as $\mathbf{M}' = \mathbf{MF}$, where $\mathbf{F} \in \mathbb{C}^{N \times N/2}$ is the $\mathsf{C2S}$ matrix. We then have $\mathbf{UM}' = \mathbf{UMF}$, for any scaled plaintext matrix $\mathbf{U} \in \mathbb{Z}^{N \times N}$. Therefore, PC-MM with slots encodings is equivalent to PC-MM with coefficients encodings when $d = N$. It may be checked that Algorithm 1 for $d > N$ are similarly compatible with slots encoding.

Combining PCMM with bootstrapping. To combine the PC-MM algorithms with bootstrapping while performing computationally demanding PC-MM at a low level, there seems to be three options:

1. $\mathsf{S2C}$, coefficients-encoded PC-MM, and then $\mathsf{HalfBTS}$;
2. Slots-encoded PC-MM, then $\mathsf{S2C}$-first bootstrapping; [$d \geq N$]
3. Slots-encoded PC-MM, then $\mathsf{C2S}$-first bootstrapping. [$d \geq N$]

In the first option, one runs PC-MM at level 1, and starts $\mathsf{S2C}$ at level $\ell_{\mathsf{S2C}} + 1$, where ℓ_{S2C} refers to the depth consumption of $\mathsf{S2C}$. This is often the most efficient option.

In the second option, one runs PC-MM at level $\ell_{\mathsf{S2C}} + 1$, and starts $\mathsf{S2C}$ at level ℓ_{S2C}. This leads to a slower PC-MM but slightly faster $\mathsf{S2C}$ for input ciphertexts, compared to the first option. The second option might be interesting if there are significantly more input ciphertexts than the output ciphertexts. For instance, for $\mathbf{U} \in \mathbb{R}^{d_1 \times d_2}$ where $d_1 \ll d_2$, this option could be preferable.

Finally, in the third option, one runs PC-MM at level 1, but starts $\mathsf{S2C}$ at higher levels. This results in fast PC-MM but slower $\mathsf{S2C}$ for the outputs. The third option might be advantageous for some limited scenarios, notably if one wants to lower the level of heavy computations, possibly unrelated to PC-MM.

MaMBo: fast PC-MM fused with batch bootstrapping. Now, we put all algorithms together, along with ring switching. We shall focus on the first option, for PC-MM between square matrices of dimension d. This framework can be extended to rectangular matrices, and similar chains can be designed for the other two options when $d \geq N$.

We are first given d^2/N slots-encoded RLWE-based FHE ciphertexts of degree N. We start with decreasing the moduli of input ciphertexts to level $\ell_{\mathsf{S2C}}+1$, and run batch $\mathsf{S2C}$ to convert them into d^2/N coefficients-encoded RLWE ciphertexts of degree N at level 1. Then, we use ring switching (see Section 2) to obtain d^2N_1/N coefficients-encoded RLWE ciphertexts of degree N_1, where N_1 is typically significantly smaller than N. This ring switching procedure provides more granularity for PC-MM, and we can select the PC-MM algorithm depending on N_1 rather than N. The subring degree N_1 should be sufficiently large for the security of the ring switching key, i.e., the parameters (N_1, PQ_1) should provide sufficient security (typical parameters are $N = 2^{16}$ and $N_1 = 2^{13}$). By evaluating the PC-MM with the algorithms of Section 4, we obtain d^2N_1/N RLWE ciphertexts of degree N_1 at level 0. Finally, we combine them into d^2/N RLWE ciphertexts of degree N at level 0 by ring switching, and use batch $\mathsf{HalfBTS}$ ($\mathsf{ModRaise}$, $\mathsf{Sh\text{-}C2S}$ and $\mathsf{EvalMod}$) to complete the overall PC-MM process.

7 Implementation and experiments

Our implementations use the HEaaN library [23] implementation of CKKS. The provided running times correspond to experiments on an Intel Xeon Gold 6242 CPU running at 2.80GHz, using a single thread. The HEaaN library takes advantage of the AVX512 instructions.

7.1 Modular PP-MM using floating-point PP-MM

Several algorithms from Section 4 reduce PC-MM to exact modular PP-MM, namely multiplication of matrices over $\mathbb{Z}_{Q_1}$. Such an exact modular matrix multiplication can be reduced to (several) floating-point matrix multiplications, allowing to take advantage of a fast BLAS implementation. This technique is folklore, and is implemented in libraries such as FFLAS-FFPACK [17, 15]. We have

re-implemented it in our settingWe refer the interested reader to the Appendix for a description of our strategy regarding this matter.

7.2 PC-MM experiments

We have implemented and experimented with the algorithms from Sections 4.1 and 4.2, which do not use the offline precomputation of the key-switching material associated to the $\mathbf{U} \cdot \mathbf{S}$ part. We take as input coefficients-encoded RLWE ciphertexts of degree 2^{13} and return output coefficients in the same format. The starting modulus $\log Q_1$ is 76, and the final modulus $\log Q_0$ is 58, where the scale factor $\Delta = 2^{42}$. We used a sparse secret key with Hamming weight 2 730. nThese parameters even allow for a switching key at a modulus PQ_1 satisfying $\log PQ_1 \leq 152$, with more than 128-bit security, based on [1]. While our algorithms can be solely used for PC-MM with bootstrapping, a switching key (towards our parameters) can bridge our parameters with FHE parameters.

As our ring degree is $N = 2^{13}$, we use MLWE format for $\log d < 13$, RLWE format for $\log d = 13$, and shared-a RLWE format for $\log d > 14$. To encrypt the $d \times d$ matrix, we took d^2/N RLWE ciphertexts as input and output, e.g., $8, 32, \cdots, 32\,768$ ciphertexts.

For the experiments, we uniformly sampled $\mathbf{U}$ and $\mathbf{M}$ in $[-1, 1]^{d \times d}$, with $\log d$ ranging from 8 to 14. We executed the PC-MM algorithms and measured the latency and the relative error between the decrypted matrix and the exact product matrix. To be more precise, if $\mathbf{X} \neq 0$ is the exact product $\mathbf{U} \cdot \mathbf{M}$ and $\mathbf{Y}$ is the result returned by our algorithm, our accuracy measure is the relative error $\|\mathbf{X}_{i,j} - \mathbf{Y}_{i,j}\|_\infty / \|\mathbf{X}_{i,j}\|_\infty$, where $\|(m_{ij})_{0 \leq i,j < d}\|_\infty$ is $\max_{0 \leq i,j < d} |m_{ij}|$.

		PP-MM			Format conv.		Accuracy		Total time	
$\log d$	Alg.	a part	b part		Pre.	Post.	b part		b part	
			(mod.)	fp.			(mod.)	fp.	(mod.)	fp.
8	Alg. 2	0.207	(7.16e-3)	4.40e-3	8.79e-3	9.00e-2	19.2/19.0	13.2/12.7	(0.315)	0.309
9	Alg. 2	0.488	(0.0308)	0.0172	0.0150	0.163	19.2/18.9	13.4/13.0	(0.696)	0.684
10	Alg. 2	1.29	(0.164)	0.0774	0.0365	0.294	19.2/19.0	13.3/12.9	(1.78)	1.70
11	Alg. 2	3.90	(0.994)	0.388	0.0811	0.721	19.2/19.0	13.4/13.0	(5.73)	5.06
12	Alg. 2	13.0	(6.60)	2.21	0.237	1.79	19.2/19.0	13.5/13.3	(21.8)	17.1
13∗		49.0	(48.3)	15.1	-	-	19.1	13.6	(96.8)	64.6
14∗	Alg. 1	186	(384)	108	33.5	20.1	19.3	13.6	(625)	347

Table 3. Experimental results for the algorithms with two PP-MM's; we report both the results with two modular PP-MM, and one modular PP-MM and one fp-PP-MM (Sec. 5). The timings are in seconds. All matrices are $d \times d$ square matrices. For rows indicated by $*$, a single experiments was run, compared for 100 for the other rows. For the accuracy column, the first figure is the worst relative accuracy, expressed in bits, whereas the second one is average relative accuracy. For $d = 2^{13}$, the (mod.) columns correspond to the LZ algorithm while the fp. columns include our floating-point optimization (Section 5).

The results are provided in Table 3. The "Format conv." columns account for the cost of converting the input to the ciphertext format required for the corresponding algorithm (i.e., the relevant parts of ModDecomp or FmtSwitch), and postprocessing to convert to the output ciphertext format (i.e., the relevant parts of ModPack or Backward-FmtSwitch). The columns "a-part" and "b-part" give the total cost of the involved PP-MM's. Finally, the "Total time" column reports both the total time for Algorithms 1 and 2 without (column "(mod.)") and with (column "fp") the optimization described in Section 5.

The figures show, in particular, a good fit with the expected complexity: the costs of handling the a-part and format conversion is expected to grow quadratically with d, whereas the cost of the b-part grows roughly as a cubic function of d. Regarding format conversion, we stress that the pre-processing phase of one algorithm reduces to simple manipulations (ModDecomp) whereas actual computations (FmtSwitch) are performed in the other algorithm; this explains the much larger figure for the pre-processing for $d = 2^{14}$. Finally, using fp-PP-MM saves a constant factor of 2 to 4; the impact of this saving, as expected, becomes noticeable as the dimension grows.

A more complete comparison with [26] can be found in Appendix D.

7.3 Batched-bootstrapping experiments

We have implemented and tested the naive version of FmtSwitch, and the algorithms Sh-S2C and Sh-C2S, and incorporated them in the batched S2C-first bootstrapping chain described in Section 6.2. We considered the *real bootstrapping* version of CKKS bootstrapping, i.e., for real-valued data. The main impact is that the EvalMod function is only evaluated once, instead of twice (for the real and imaginary parts) in the case of complex bootstrapping. We provide information about the HEaaN library parameters set we used, in Table 4

Table 5 gives our experimental results. The first column (single ciphertext case) is the reference timing for the current bootstrapping implementation in HEaaN. We have experimented batches of up to $k = 32$ bootstrappings at a time. These experiments demonstrate an improvement by a factor > 3 on the S2C+C2S part and a speedup of a factor 2 over the whole bootstrapping chain.

N	$\log_2(QP)$	$\log_2(Q)$	dnum	depth
2^{16}	1 555	1 258	5	9

Table 4. Information on the parameter preset (FGb) of the HEaaN library we used to test batch bootstrapping. dnum denotes the gadget rank.

7.4 MaMBo cost

Finally, even though we have not implemented MaMBo as a whole, we have measured the timings for all individual MaMBo steps. We thus can estimate the

Batch size	1	2	4	8	16	32
FmtSwitch	-	0.039	0.146	0.426	0.994	2.39
Sh-S2C	0.89	1.20	1.85	3.17	5.86	11.4
Sh-C2S	7.44	9.55	13.7	21.3	37.6	71.5
Backward-FmtSwitch	-	0.251	0.406	0.7	1.34	2.54
EvalMod	2.60	5.16	10.4	20.8	42.2	84.5
Full BTS	11.7	17.5	28.6	50.3	95.4	187
Amortized BTS	11.7	8.75	7.15	6.29	5.96	5.84
S2C+C2S speedup	-	1.55	2.14	2.72	3.07	3.21
Total speedup	-	1.33	1.63	1.86	1.96	2.00

Table 5. Timings (in seconds) for our implementation of S2C-first batched bootstrapping for real inputs (see Section 6.2).

total cost of MaMBo the fused PC-MM with batch bootstrapping. We limit ourselves to Option 1 of Section 6.3. We recall the steps of MaMBo below. The indices outside of the brackets refer to the level(s) where each step is executed.

$$[\mathsf{FmtSwitch}]_4 \to [\mathsf{Sh\text{-}S2C}]_{4\to 1} \to [\mathsf{RingSwitch}_{2^{16}\to 2^{13}}]_1 \to [\text{PC-MM}]_{1\to 0}$$
$$\to [\mathsf{RingSwitch}_{2^{13}\to 2^{16}}]_0 \to [\mathsf{FmtSwitch}]_0 \to [\mathsf{HalfBTS}] \ .$$

For the sake of comparison, HEaaN's current implementation of the complex bootstrapping algorithm (starting at level 3) costs 15.3s per ciphertext in our experimental setting; for large matrices, MaMBo is thus 41% faster than the current bootstrapping (without PC-MM).

d	FmtSwitch (lev. 4)	Sh-S2C (lev. 4)	RingSwitch Fwd.	PC-MM (lev. 1)	RingSwitch Bck.	FmtSwitch (lev. 0)	HalfBTS	MaMBo total	MaMBo /ctxt
2^8	-	1.20	0.0196	0.361	0.0153	-	16.4	18.0	18.0
2^9	0.151	2.63	0.0497	0.735	0.0535	0.0513	38.6	42.3	10.6
2^{10}	0.888	8.44	0.183	1.82	0.212	0.474	131	143	8.92
2^{11}	6.23	31.8	0.700	5.16	0.786	2.94	533	581	9.07
2^{12}	22.2	121	2.85	17.9	3.20	11.8	2130	2310	9.03
2^{13}	104	551	12.3	65.3	13.3	51.4	8410	9200	8.99
2^{14}	432	1970	47.8	355	54.2	192	34200	34200	9.09

Table 6. MaMBo cost as a function of the dimension. All timings are given in seconds. For the bootstrapping process, real ciphertexts are pairwise combined into complex ciphertexts, and complex bootstrapping is used. A single experiment was run for all d.

This table shows that MaMBo quickly attains an asymptotic regime of $9s$ per ciphertext for fused PC-MM and bootstrapping (as soon as the dimension exceeds 2^{10}). Our interpretation is that the variations around this value are related to noise in timings measurements, which might be linked to the huge memory footprint of PC-MM in those dimensions.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. (2015), software available at `https://github.com/malb/lattice-estimator`, git commit# 5350825
2. Bae, Y., Cheon, J.H., Kim, J., Park, J.H., Stehlé, D.: HERMES: efficient ring packing using MLWE ciphertexts and application to transciphering. In: CRYPTO (2023)
3. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM. In: EuroS&P (2018)
4. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: CHIMERA: combining Ring-LWE-based fully homomorphic encryption schemes. J. Math. Cryptol. (2020)
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapSVP. In: CRYPTO (2012)
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory (2014)
7. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: STOC (2013)
8. Chen, H., Dai, W., Kim, M., Song, Y.: Homomorphic conversion between (ring) LWE ciphertexts. In: ACNS (2021)
9. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: EUROCRYPT (2018)
10. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: ASIACRYPT (2017)
11. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In: ASIACRYPT (2017)
12. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: CSCML (2021)
13. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding (2018), available at `https://arxiv.org/abs/1810.04805`
14. Ding, Y., Guo, H., Guan, Y., Liu, W., Huo, J., Guan, Z., Zhang, X.: East: Efficient and accurate secure transformer framework for inference (2023), available at `https://arxiv.org/abs/2308.09923`
15. Dumas, J.G., Giorgi, P., Pernet, C.: Dense linear algebra over word-size prime fields: the FFLAS and FFPACK packages. ACM Trans. on Mathematical Software (2008)
16. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption (2012), available at `http://eprint.iacr.org/2012/144`
17. FFLAS14, T.F.F.G.: FFLAS-FFPACK: Finite Field Linear Algebra Subroutines / Package, v2.0.0 edn. (2014), `http://linalg.org/projects/fflas-ffpack`
18. Froelicher, D., Cho, H., Edupalli, M., Sousa, J.S., Bossuat, J.P., Pyrgelis, A., Troncoso-Pastoriza, J.R., Berger, B., Hubaux, J.P.: Scalable and privacy-preserving federated principal component analysis (2023), available at `https://arxiv.org/abs/2304.00129`
19. Gentry, C., Halevi, S., Peikert, C., Smart, N.P.: Ring switching in BGV-style homomorphic encryption. In: SCN (2012)
20. Goldwasser, S., Kalai, Y.T., Peikert, C., Vaikuntanathan, V.: Robustness of the learning with errors assumption. In: ICS (2010)

21. Halevi, S., Shoup, V.: Algorithms in HElib. In: CRYPTO (2014)
22. Hao, M., Li, H., Chen, H., Xing, P., Xu, G., Zhang, T.: Iron: Private inference on transformers. Advances in Neural Information Processing Systems (2022)
23. HEaaN, C.: HEaaN library (2022), available at https://www.cryptolab.co.kr/en/products-en/heaan-he/
24. Jiang, X., Kim, M., Lauter, K., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: CCS (2018)
25. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Des. Codes Cryptogr. (2015)
26. Liu, J., Zhang, L.F.: Privacy-preserving and publicly verifiable matrix multiplication. IEEE Transactions on Services Computing (2022)
27. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT (2010)
28. OpenBLAS: An optimized BLAS library – version 0.3.26, available at https://www.openblas.net/
29. Pang, Q., Zhu, J., Mollering, H., Zheng, W., Schneider, T.: BOLT: Privacy-preserving, accurate and efficient inference for transformers (2023), available at https://eprint.iacr.org/2023/1893
30. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: STOC (2009)
31. Peikert, C., Waters, B.: Lossy trapdoor functions and their applications. In: STOC (2008)
32. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training (2018), available at https://openai.com/research/language-unsupervised
33. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: ASIACRYPT (2009)
34. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., Lample, G.: LLaMA: Open and efficient foundation language models (2023), available at https://arxiv.org/abs/2302.13971
35. Zhang, J., Liu, J., Yang, X., Wang, Y., Chen, K., Hou, X., Ren, K., Yang, X.: Secure transformer inference made non-interactive (2023), available at https://eprint.iacr.org/2024/136

Appendices

A On sub-rings and MLWE-RLWE conversions

For the sake of completeness, we describe the procedures mentioned in 2.3.

Subrings. We define the cyclotomic ring $\mathcal{R}_{Q,N'}$ of order $2N'$ as $\mathcal{R}_{Q,N'} = \mathbb{Z}_Q[Y]/(Y^{N'}+1)$. The map $Y \mapsto X^{N/N'}$ extends to a ring-homomorphism from $\mathcal{R}_{Q,N'}$ to $\mathcal{R}_{Q,N}$, which embeds $\mathcal{R}_{Q,N'}$ into $\mathcal{R}_{Q,N}$. The ring $\mathcal{R}_{Q,N}$ contains k copies of $\mathcal{R}_{Q,N'}$: an element $a = a_0 + a_1 X + \ldots + a_{N-1}X^{N-1} \in \mathcal{R}_{Q,N}$ may be viewed as k elements in $\mathcal{R}_{Q,N'}$ by writing

$$\begin{aligned} a = &\left(a_0 + a_k X^k + \ldots + a_{k(N'-1)} X^{k(N'-1)}\right) \\ &+ X\left(a_1 + a_{k+1}X^k + \ldots + a_{k(N'-1)+1}X^{k(N'-1)}\right) \\ &+ \ldots \\ &+ X^{k-1}\left(a_{k-1} + a_{2k-1}X^k + \ldots + a_{kN'-1}X^{k(N'-1)}\right) \ , \end{aligned} \tag{14}$$

and identifying X^k with the indeterminate Y of $\mathcal{R}_{Q,N'}$. For an element $a \in \mathcal{R}_{Q,N}$, we shall use the notation $\mathrm{e}_i^*(a)$ for the coefficient of X^i in the decomposition (14) of a over $\mathcal{R}_{Q,N'}$.

Ring-switching. The goal is to decompose a ciphertext $\mathsf{ct} = (a,b) \in (\mathcal{R}_{Q,N})^2$ for $\mathsf{RLWE}_{Q,N}$ under a key $\mathsf{sk} \in \mathcal{R}_N$ to a set of ciphertexts $(\mathsf{ct}'_i = (a'_i, b'_i))_{0 \le i < k} \in (\mathcal{R}^2_{Q,N'})^k$ for $\mathsf{RLWE}_{Q,N'}$ under a secret key $\mathsf{sk}' \in \mathcal{R}_{N'}$. This is accomplished by first switching key from sk to sk' using a switching key

$$\mathsf{swk}_{Q,P,\mathsf{sk}\to\mathsf{sk}'} = (a_{\mathsf{swk}}, b_{\mathsf{swk}}) = (a_{\mathsf{swk}}, -a_{\mathsf{swk}} \cdot \mathsf{sk}' + e + P \cdot \mathsf{sk}) \in \mathcal{R}^2_{PQ,N} \ ,$$

for which sk' belongs to the subring $\mathcal{R}_{N'}$, i.e., whose non-zero coefficients as viewed as an element of $\mathcal{R}_N$ can only correspond to the X^{ki}'s for $0 \le i < N'$. In fact, the pair $(a_{\mathrm{swk}}, b_{\mathrm{swk}})$ may be viewed as k RLWE pairs in the subring $\mathcal{R}_{PQ,N'}$ that share the same secret. Note that providing this switching key requires the security of RLWE in degree N', which in turns requires that PQ should not be too large. Ring-switching is then described as:

- $\mathsf{RingSwitch}$. On input $\mathsf{ct} = (a,b) \in \mathcal{R}_{Q,N}$ and $\mathsf{swk}_{Q,P,\mathsf{sk}\to\mathsf{sk}'}$, set $(\widetilde{a}, \widetilde{b}) \leftarrow \mathsf{KeySwitch}((a,b), \mathsf{swk}_{Q,P,\mathsf{sk}\to\mathsf{sk}'})$, and return $(a_i, b_i) = (\mathrm{e}_i^*(\widetilde{a}), \mathrm{e}_i^*(\widetilde{b}))$ for $0 \le i < k$.

From RLWE to MLWE. Let $\mathsf{ct} = (a,b) \in \mathcal{R}^2_{Q,N}$ be an $\mathsf{RLWE}_{Q,N}$ ciphertext under a secret key $\mathsf{sk} \in \mathcal{R}_N$. The identity $a \cdot \mathsf{sk} + b = m$ may be rewritten in module form as

$$\begin{aligned} \sum_{0 \le i < k} \mathrm{e}_i^*(m) X^i &= \sum_{0 \le j,\ell < k} \mathrm{e}_\ell^*(a) X^\ell \mathrm{e}_j^*(\mathsf{sk}) X^j + \sum_{0 \le i < k} \mathrm{e}_i^*(b) X^i \\ &= \sum_{0 \le i < k} \left(\langle \widetilde{\mathbf{a}}_i, \mathsf{sk}' \rangle + \mathrm{e}_i^*(b)\right) X^i \ , \end{aligned} \tag{15}$$

where $\mathbf{sk}' = (\mathrm{e}_0^*(\mathsf{sk}), \ldots, \mathrm{e}_{k-1}^*(\mathsf{sk})) \in \mathcal{R}_{N'}^k$ and

$$\widetilde{\mathbf{a}}_i = (\mathrm{e}_i^*(a), \mathrm{e}_{i-1}^*(a), \ldots, \mathrm{e}_0^*(a), Y\mathrm{e}_{k-1}^*(a), \ldots, Y\mathrm{e}_{i+1}^*(a)) \in \mathcal{R}_{Q,N'}^k \ .$$

This shows that for $0 \leq i < k$, the tuple $(\widetilde{\mathbf{a}}_i, \mathrm{e}_i^*(b))$ is an $\mathsf{MLWE}_{Q,N'}^k$ encryption of m_i under the key $\mathbf{sk}'$. We thus define the RLWE to MLWE decomposition as:

- ModDecomp. On input $\mathsf{ct} = (a, b)$, return the k ciphertexts $(\widetilde{\mathbf{a}}_i, \mathrm{e}_i^*(b))$ for $0 \leq i < k$, defined as above.

As a and sk play a symmetric role in Equation (15), we can also write

$$\sum_{0 \leq i < k} \mathrm{e}_i^*(m) X^i = \sum_{0 \leq i < k} \left(\langle \mathbf{a}', \widetilde{\mathbf{sk}}_i \rangle + \mathrm{e}_i^*(b) \right) X^i \ ,$$

where $\mathbf{a}'$ and $\widetilde{\mathbf{sk}}$ are obtained from $\mathbf{a}$ and $\mathbf{sk}$ as in the previous case. We then obtain a decomposition in k ciphertexts with common a part, but distinct secret keys; this gives the ShModDecomp algorithm.

From MLWE to RLWE. Let $\mathbf{ct} = (\mathbf{a}, b) \in \mathcal{R}_{Q,N'}^{k+1}$ be an MLWE-ciphertext for $\mathsf{sk} \in \mathcal{R}_{N'}^k$. We define $\widetilde{\mathsf{sk}} = \sum_{0 \leq i < k} \mathsf{sk}_i X^i \in \mathcal{R}_N$ and let $\mathsf{swk}_{P,Q,\widetilde{\mathsf{sk}} \to \mathsf{sk}'} \in \mathcal{R}_{PQ,N}$ be an RLWE-switching key from $\widetilde{\mathsf{sk}}$ to a new key sk'.

- ModKeySwitch goes through the following steps:
  1. Embed: $\widetilde{a} \leftarrow \mathbf{a}_0 + \sum_{1 \leq i < k} \mathbf{a}_i X^i Y^{-1} \in \mathcal{R}_{Q,N}$; then $(\widetilde{a}, b)$ is an $\mathsf{RLWE}_{Q,N}$ encryption of $\widetilde{m} \in \mathcal{R}_{Q,N}$ such that $\mathrm{e}_0^*(\widetilde{m}) = m$.
  2. $(a', b') \leftarrow \mathsf{KeySwitch}((\widetilde{a}, b), \mathsf{swk}_{Q,P,\widetilde{\mathsf{sk}} \to \mathsf{sk}'})$.
  3. Extract: using ModDecomp, recover $(\widetilde{\mathbf{a}}'_0, b'_0)$ encrypting $\mathrm{e}_0^*(\widetilde{m}) = m$ under $(\mathrm{e}_0^*(\mathsf{sk}'), \ldots, \mathrm{e}_{k-1}^*(\mathsf{sk}'))$.

Module packing takes as input $(\mathsf{ct}_i)_{0 \leq i < k}$ where $\mathsf{ct}_i = (\mathbf{a}_i, b_i)$ encrypts m_i under a common MLWE-secret key $\mathbf{sk} \in \mathcal{R}_{N'}^k$. It first combines the various ct_i's in polynomial form, turning the $k = N/N'$ $\mathsf{MLWE}_{Q,N'}^k$ ciphertexts into a single $\mathsf{MLWE}_{Q,N}^k$ ciphertext, then reduces the MLWE dimension to 1 by switching, separately, all the components of the MLWE secret key vector to sk'. It thus requires switching keys $\mathsf{swk}_i = \mathsf{swk}_{Q,P,\mathbf{sk}_i \to \mathsf{sk}'}$. It outputs a single $\mathsf{RLWE}_{Q,N}$ ciphertext.

- ModPack proceeds as follows:
  1. $\mathsf{ct}' = (\mathbf{A}, B) \leftarrow (\sum_{0 \leq i < k} \mathbf{a}'_i X^i, \sum_{0 \leq i < k} b'_i X^i) \in \mathcal{R}_{Q,N}^{k+1}$.
  2. For $0 \leq i < k$, run $(\mathbf{A}'_i, \beta_i) \leftarrow \mathsf{KeySwitch}((\mathbf{A}_i, 0), \mathsf{swk}_i)$.
  3. Return $(\sum_{0 \leq i < k} \mathbf{A}'_i, B + \sum_{0 \leq i < k} \beta_i) \in \mathcal{R}_{Q,N}^2$, which encrypts $m = \sum_{0 \leq i < k} m_i X^i$ under sk'.

The correctness of the process follows from the identity $m - B \approx \sum_{0 \leq i < k} \mathbf{A}_i \cdot \mathbf{sk}_i \approx \sum_{0 \leq i < k} (\mathbf{A}'_i \cdot \mathsf{sk}' + \beta_i)$.

Algorithm 3 Precomputation-based PC-MM for dimension above the RLWE ring degree N

Input: A matrix $\mathbf{U} \in \mathbb{R}^{d\times d}$, with $d = kN$ for some integer $k \geq 1$,
coefficients-encoded RLWE ciphertexts $(\mathsf{ct}_i)_{0\leq i<d^2/N}$ in $\mathcal{R}_{Q_1,N}$,
each of which encrypts a segment of a row of a matrix $\mathbf{M} \in \mathbb{R}^{d\times d}$.
Input: Switching keys $\mathsf{fmt\text{-}swk}_{Q_1,P,\mathsf{sk}\to\{\mathsf{sk}_i\}_{0\leq i<k}}$ and $\mathsf{swk}_{Q_0,P,\mathsf{sk}'_i\to\mathsf{sk}}$ for $0 \leq i < k$,
where $\mathbf{U}[\mathsf{sk}_0^T|\cdots|\mathsf{sk}_{d-1}^T]^T = [{\mathsf{sk}'_0}^T|\cdots|{\mathsf{sk}'_{d-1}}^T]^T$.
Output: Coefficients-encoded RLWE ciphertexts in $\mathcal{R}_{Q_0,N}$,
each of which encrypts a segment of a row of the matrix $\mathbf{UM} \in \mathbb{R}^{d\times d}$.

for $i \leftarrow 0$ **to** d **do**
  $(a_i, b_{i,j}) \leftarrow \mathsf{FmtSwitch}(\mathsf{ct}_{ki}, \cdots, \mathsf{ct}_{k(i+1)-1}; \mathsf{fmt\text{-}swk}_{Q_1,P,\mathsf{sk}\to\{\mathsf{sk}_i\}_{0\leq i<k}})$
end for
$\mathbf{B} \leftarrow (b_{i,j})_{0\leq i<d,0\leq j<d}$
$\mathbf{B}' \leftarrow \mathbf{U}\cdot\mathbf{B}$
for $i \leftarrow 0$ **to** d **and** $j \leftarrow 0$ **to** k **do**
  $\mathsf{ct}'_{i,j} \leftarrow (\sum_\ell a_{i,\ell}X^\ell, \sum_\ell b'_{i,jN+\ell}X^\ell) \in R^2_{Q_1,N}$
  $\mathsf{ct}'_{i,j} \leftarrow \mathsf{KeySwitch}(\mathsf{ct}'_{i,j}; \mathsf{swk}_{Q_0,P,\mathsf{sk}'_i\to\mathsf{sk}})$
  $\mathsf{ct}'_{i,j} \leftarrow \mathsf{Rescale}(\mathsf{ct}'_{i,j}; Q_1, Q_0)$
end for
return all $(\mathsf{ct}')_{i,j}$'s

B Precomputation-based algorithms

In this Section, we give a full description of the precomputation-based algorithms outlined in Sections 4.3. Algorithm 3 is to be used when the dimension d is larger than the ring degree N, whereas Algorithm 4 is to be used for $d < N$.

C Reduction of Modular PP-MM to floating-point PP-MM

For $d < 2^{15}$, we have implemented our own reduction from one PP-MM between a matrix $\mathbf{U}$ with entries bounded by 2^{18} and a matrix modulo $q_0 \approx 2^{58}$, to three floating-point PC-MM's using BLAS plus a quadratic (in d) number of integer operations.

We shall not undertake a systematic study of this reduction, which is folklore; we restrict ourselves to the parameter values of our implementation. We wish to compute $U \cdot M$, where M is defined modulo $q_0 < 2^{60}$, and the coefficients of U are bounded by $\|U\|_\infty = 2^{18}$. For these parameter values, we claim that for $d < 2^{15}$, PP-MM modulo q_0 can be reduced to 3 floating-point PP-MM, plus a quadratic (in d) number of operations on integers $\leq 2^{93}$ in absolute value.

We define $\delta = 2^{20}$, and given an integer $x \in [0, q_0 - 1]$, we write $x = x^{(2)}\cdot\delta^2 + x^{(1)}\cdot\delta + x^{(0)}$ the base-δ decomposition of x, where $0 \leq x_i < \delta, i = 0, 1, 2$. Given a $d \times d$ matrix $M = (\tilde{m}_{ij})_{0\leq i,j<d}$ over $\mathbb{Z}/q_0\mathbb{Z}$, and letting m_{ij} denote the unique

Algorithm 4 Precomputation-based PC-MM for dimension below the RLWE ring degree N

Input: A matrix $\mathbf{U} \in \mathbb{R}^{d \times d}$, with $N = kd$ for some integer $k \geq 1$,
coefficients-encoded RLWE ciphertexts $(\mathsf{ct}_i)_{0 \leq i < d^2/N}$ in $\mathcal{R}_{Q_1,N}$,
each of which encrypts a strip of rows of a matrix $\mathbf{M} \in \mathbb{R}^{d \times d}$.
Input: Switching keys $\mathsf{fmt\text{-}swk}_{Q_1,P,\mathsf{sk} \to \{\mathsf{sk}_i\}_{0 \leq i < d^2/N}}$, $\mathsf{swk}_{Q_0,P,\mathsf{sk}'_i \to \mathsf{sk}}$ for $0 \leq i < k$,
where $\mathbf{U}[\mathsf{sk}_0^T | \cdots | \mathsf{sk}_{d-1}^T]^T = [\mathsf{sk}_0'^T | \cdots | \mathsf{sk}_{d-1}'^T]^T$, and switching keys for $\mathsf{ModPack}$
Output: Coefficients-encoded RLWE ciphertexts in $\mathcal{R}_{Q_0,N}$,
each of which encrypts a strip of rows of the matrix $\mathbf{UM} \in \mathbb{R}^{d \times d}$.

$(a, b_i) \leftarrow \mathsf{FmtSwitch}(\mathsf{ct}_i)_{0 \leq i < d^2/N}; \mathsf{fmt\text{-}swk}_{Q_1,P,\mathsf{sk} \to \{\mathsf{sk}_i\}_{0 \leq i < d^2/N}})$
for $i \leftarrow 0$ **to** d^2/N **do**
  $((\mathbf{a}_{di+j}, b_{di+j}))_{0 \leq j < k} \leftarrow \mathsf{ShModDecomp}(\mathsf{ct}_i)$
end for
for $i \leftarrow 0$ **to** d **do**
  Set the i-th row of $\mathbf{B}$ as the coefficients of b_i
end for
$\mathbf{B}' \leftarrow \mathbf{UB}$
for $i \leftarrow 0$ **to** d **do**
  Set the coefficients of b'_i as the i-th row of $\mathbf{B}'$
end for
for $i \leftarrow 0$ **to** d^2/N **do**
  for $j \leftarrow 0$ **to** k **do**
    $(a'_{di+j}, b'_{di+j}) \leftarrow \mathsf{KeySwitch}((a'_{di+j}, b'_{di+j}); \mathsf{swk}_{Q_0,P,\mathsf{sk}'_i \to \mathsf{sk}})$
  end for
  $\mathsf{ct}'_i \leftarrow \mathsf{ModPack}((a'_{di+j}, b'_{di+j})_{0 \leq j < k})$
  Rescale $(\mathsf{ct}'_i; Q_1, Q_0)$
end for
return all ct'_i's

integer congruent to $\tilde{m}_{ij}$ modulo q, we extend this notation to the matrix M as $M = \delta^2 \cdot M^{(2)} + \delta \cdot M^{(1)} + M^{(0)}$, where $M^{(k)} = (m_{ij}^{(k)})_{0 \le i,j < d}$ for $k = 0, 1, 2$.

The largest integer occuring in the computation of $U \cdot M^{(i)}$, $i = 0, 1, 2$, is upper bounded by $d\|U\|_\infty \delta < 2^{53}$ in absolute value. We can thus compute exactly each of the $U \cdot M^{(i)}$, using floating-point arithmetic; the sum

$$\delta^2 \cdot UM^{(2)} + \delta \cdot UM^{(1)} + UM^{(0)}, \tag{16}$$

which has quadratic cost (smaller than matrix multiplication for d large enough) is then evaluated using integer arithmetic. Finally, one can check that the integers occuring in this linear combination are $< 2^{93}$ in absolute value, as claimed.

D Comparison to [26]

We conducted experiments to compare our algorithms with optimization in Section 5 to the LZ algorithm introduced in [26], under the same parameters in Section 7.2. We note that all parameters achieve 128-bits security. Since [26] does not provide experimental results for PC-MM, in order to provide a fair comparison, we implemented [26] for $d = N$ by using OpenBLAS [28] as our algorithms.

However, the [26] algorithm, as described, is restricted to a matrix dimension d equal to, or larger than, the RLWE ring degree N. One trivial modification is to use fewer RLWE slots; that is, in order to encrypt a $d \times d$ matrix, we allocate d RLWE ciphertexts with d messages out of N available slots. We have implemented this variant of [26] algorithm for small matrices, in order to be able to provide a comparison with our algorithms addressing a larger range of dimensions. However, we remark that this modification might incur additional computation and communication costs before and after PC-MM.

For all cases, our optimization in Section 5 improves the efficiency a lot, and the cost of format conversions is relatively minor compared to the improvement from our optimization. We remark that our LZ implementation might have larger precision (e.g., ≈ 19.2 bits) since we did not use the optimization technique in Section 5, while our algorithms also provide reasonable precision (e.g., ≈ 13.4). Table 7 reports the timings in seconds.

$\log d$	8	9	10	11	12	13	14
[26]	0.423	1.05	2.75	8.13	27.4	96.8	786
Ours	0.309	0.684	1.70	5.06	17.1	64.6	347

Table 7. Comparison between our algorithms and our implementation of [26]. All timings are in seconds, for the product of two $d \times d$ matrices.

E Table of notations

PC-MM, PP-MM	Sec. 1, p .1	(FP, Mod)-PP-MM	Table 1, p. 3
RLWE	Sec. 2.1, p. 8	MLWE	Sec. 2.1, p. 8
$\mathcal{R}, \mathcal{R}_N, \mathcal{R}_q, \mathcal{R}_{q,N}$	Sec. 2, p. 8	ω	Sec. 2, p. 8
N	RLWE ring-degree	d	input matrix dimension
k	MLWE rank	ℓ_{S2C}	Sec. 2.2, p. 11
Q_i	Sec. 2.2, p. 10	Q_{top}	Sec. 2.2, p. 10
Toep()	Sec. 1.1, p. 4	$a_{i,[j,k)}$	Sec. 4.1, p. 17

DFT, iDFT	Sec. 2.2, p. 9	S2C, C2S, ModRaise	Sec. 2.2, p. 11
$\mathsf{Ecd}_{\mathrm{coeff}}$	Sec. 2.2, p. 9	$\mathsf{Ecd}_{\mathrm{slot}}$, $\mathsf{Dcd}_{\mathrm{slot}}$	Sec. 2.2, p. 9
FmtSwitch	Sec. 3.2, p. 13	Backward-FmtSwitch	Sec. 3.4, p. 15
ModDecomp, ShModDecomp	Sec. 2.3, p. 12	ModPack	Sec. 2.3, p. 12
EvalMod	Sec. 2.2, p. 11	Sh-S2C, Sh-C2S	Sec. 6.2, p. 27
KeySwitch	Sec. 2.2, p. 9	ModKeySwitch	Sec. 2.3, p. 12
HalfBTS	Sec. 2.2, p. 11	Rescale	Sec. 2.2, p. 10