Fast Secure Matrix Multiplications over Ring-Based Homomorphic Encryption

Pradeep Kumar Mishra[1], Deevashwer Rathee[2], Dung Hoang Duong[3], and Masaya Yasuda[4]

[1] Graduate School of Mathematics, Kyushu University,
744 Motooka Nishi-ku, Fukuoka 819-0395, Japan
p-mishra@math.kyushu-u.ac.jp
[2] Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi 221005, India.
deevashwer.student.cse15@iitbhu.ac.in
[3] School of Computing and Information Technology, University of Wollongong,
Northfields Avenue, Wollongong NSW 2522, Australia.
hduong@uow.edu.au
[4] Institute of Mathematics for Industry, Kyushu University,
744 Motooka Nishi-ku, Fukuoka 819-0395, Japan
yasuda@imi.kyushu-u.ac.jp

Abstract. Secure matrix computation is one of the most fundamental and useful operations for statistical analysis and machine learning with protecting the confidentiality of input data. Secure computation can be achieved by homomorphic encryption, supporting meaningful operations over encrypted data. HElib is a software library that implements the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic scheme, in which secure matrix-vector multiplication is proposed for operating matrices. Recently, Duong et al. (Tatra Mt. Publ. 2016) proposed a new method for secure single matrix multiplication over a ring-LWE-based scheme. In this paper, we generalize Duong et al.'s method for secure *multiple* matrix multiplications over the BGV scheme. We also implement our method using HElib and show that our method is much faster than the matrix-vector multiplication in HElib for secure matrix multiplications.

Key words: Secure matrix multiplications, Leveled fully homomorphic encryption, Packing methods.

1 Introduction

Recent development of cloud computing allows users to easily outsource their data in the cloud. On the other hand, security and privacy concerns for stored data in the cloud have risen. An excellent solution for the issue is to store all the data in encrypted format and perform operations on encrypted data. Homomorphic encryption can achieve the solution, and hence it has been expected as a powerful tool in cloud computing. The concept of homomorphic encryption was

first introduced by Rivest et al. in 1978 [25]. About 30 years later, the first scheme of *fully homomorphic encryption (FHE)* that supports arbitrary operations on encrypted data was constructed by Gentry [11]. After Gentry's pioneering work, a number of FHE schemes have been proposed and improved in both theory and practice. However, currently known FHE schemes are yet impractical for real applications. For example, it is reported in a recent work [7] that it takes 52 milliseconds for *single gate* bootstrapping to refresh the error in a ciphertext for unlimited operations. On the other hand, the Paillier scheme [24] and the BGN scheme [2] are practical, but their functionality is very limited. In recent years, *somewhat homomorphic encryption (SHE)*, initially used as a building block for FHE construction, and its leveled improvement (called *leveled FHE*) have attracted a lot of attention from various communities. Both SHE and leveled FHE can support only a limited number of additions and multiplications, but it is applicable in various scenarios with reasonable performance (e.g., see [23, 13, 29, 1, 31, 6]). In particular, since leveled FHE has slower growth of the error in a ciphertext, it is more useful to evaluate a circuit of large depth.

Matrix computation is one of the most basic and useful operations for various applications, including statistical analysis, image processing and machine learning. In this paper, we focus on secure matrix multiplications (see Figure 1 below for an image of our goal). At present, ring-based leveled FHE schemes, such as BGV [4], FV [10], YASHE [3], and NTRU [19], are efficient and useful. BGV and FV schemes are based on the ring-LWE (learning with errors) assumption [21], and YASHE is a variant of NTRU. Costache and Smart [8] compared features of such schemes (see also [18] for a comparison), and showed that the BGV scheme is more efficient for large plaintext space than other schemes. Hence we use `HElib` [16] as a software library in our implementation of the BGV scheme. (Recently, `HElib` has been improved for efficiency [17].) For secure matrix computation, matrix-vector multiplication is proposed in `HElib` (see [15] for its manual). Recently, Lu et al. [20] slightly modified the matrix-vector multiplication for secure statistical analysis over `HElib`. As an individual work, Duong et al. [9] proposed efficient methods for secure *single* matrix multiplication over ring-based SHE schemes. Later, Wang et al. [28] modified Duong et al.'s methods for flexible matrix computation, but their modification is much less efficient for matrices of larger size.

Our Contributions We generalize Duong et al.'s methods [9] for secure *multiple* matrix multiplications over the BGV scheme using $R = \mathbb{Z}[x]/(x^n + 1)$ as the base ring for a 2-power integer n. Our method can be used over the other ring-based SHE schemes. For secure multiplication between two matrices $\mathbf{A}$ and $\mathbf{B}$, a main ingredient of [9] is to pack $\mathbf{A}$ and $\mathbf{B}$ into two types of polynomial over R, and then encrypt the polynomials. The homomorphic property of the *native* plaintext space of the BGV scheme enables us to compute all the entries of $\mathbf{A} \times \mathbf{B}$ over packed ciphertexts by homomorphic multiplication. Our new strategy for multiple matrix multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ is to propose three types of polynomial in R for $\mathbf{A}_1 \times \mathbf{A}_2$, $\mathbf{A}_3 \times \cdots \times \mathbf{A}_{\ell-1}$, and $\mathbf{A}_\ell$. Specifically, we pack

$\mathbf{A}_1$ by the first transformation of [9], and flip the columns of $\mathbf{A}_2$ and pack them using a method similar to $\mathbf{A}_1$ to obtain the entries of $\mathbf{A}_1 \times \mathbf{A}_2$ in polynomial format. We propose new transformation to pack the product $\mathbf{A}_3 \times \cdots \times \mathbf{A}_{\ell-1}$ into polynomial format. In contrast, we make use of the second transformation of [9] for $\mathbf{A}_\ell$. But our method requires for us to take an appropriate jump for exponents of the variable x of the polynomial ring R greater than the total degree of polynomials over R corresponding to the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_{\ell-1}$, in order to avoid overlapping the coefficients of the decryption polynomial.

Different from ours, the matrix-vector multiplication implemented in `HElib` makes use of the plaintext *slots* of the BGV scheme, useful for SIMD (Single Instruction Multiple Data) operations (see [13] for a typical use). While the matrix-vector multiplication can only pack a row or column vector into a single ciphertext, ours can pack a matrix. Our method can also perform secure matrix multiplications by a few homomorphic multiplications over our packed ciphertexts. For a comparison of performance, we implement our method and compare with the matrix-vector multiplication over `HElib`. With our implementation results, we also discuss advantages and limitations of our method.

This is a fully revised paper of the conference paper [22]. Main differences are as follows:

1. We complete a proof of our secure matrix multiplications among three matrices (Theorem 2 below).
2. We also give a generalization of Duong et al.'s method for $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ with $\ell \geq 4$ (Subsection 3.2 below).
3. While the BV scheme [5] is implemented in [22], we here implement the BGV scheme over `HElib` for our secure matrix multiplications. Furthermore, we compare performance results of our method and the matrix-vector multiplication over `HElib` (Section 4 below).

2 Preliminaries

`HElib` [16] is an open-source `C++` library that implements a variant of the ring-LWE-based BGV scheme [4], improved by Gentry, Halevi and Smart [12]. It includes the Smart-Vercauteren ciphertext packing [26], high-level procedures for data-movement and simple linear algebra, modulus and key switching operations, and bootstrapping. In this section, we review the BGV scheme implemented in `HElib`.

2.1 The BGV Scheme

The BGV scheme is a leveled homomorphic encryption scheme based on ring-LWE. For a positive integer N, the scheme is defined over the ring

$$R = \mathbb{Z}[x]/(\Phi_N(x)),$$

where $\Phi_N(x)$ is the N-th cyclotomic polynomial of degree $\phi(N)$ (here ϕ is the Euler's totient function). Every polynomial in R can be represented as a vector

with coefficients in $\mathbb{Z}$ (*coefficient representation*). A distribution χ over R is implemented in `HElib` for error polynomials, drawing a random polynomial in R with every coefficient chosen at random from a discrete Gaussian distribution with zero mean and variance σ^2 for a parameter σ ($\sigma = 3.2$ is default in `HElib`). For an odd prime q, denote by $[\cdot]_q$ the reduction modulo q, mapping an element of R to the element of R with every coefficient equal to the unique representative of its equivalence class modulo q in the interval $(-q/2, q/2]$. For a prime or prime-power p, we denote the ring $R_p = R/pR$.

The *native* plaintext space of the BGV scheme, implemented in `HElib`, is R_t for a prime-power $t = p^r$ (cf., see Subsection 2.3 below for plaintext slots). The scheme is parametrized by a sequence of decreasing moduli

$$q_L > q_{L-1} > \cdots > q_0 \tag{1}$$

for homomorphic evaluation. For every modulus q_i, an i-th level ciphertext in the scheme is a vector in $R_{q_i}^2$. A secret key in `HElib` is chosen as an element $s \in R$ with coefficients in $\{0, \pm 1\}$ and low Hamming weight H (e.g., $H = 64$). Set $\mathsf{sk} = (1, -s)$ as a secret key. For correct decryption, every i-th level ciphertext $v_i = (c_0, c_1) \in R_{q_i}^2$ of a plaintext $\alpha \in R_t$ should satisfy that the "noise" polynomial $[\langle \mathsf{sk}, v_i \rangle]_{q_i} = [c_0 - c_1 s]_{q_i} \in R$ has the form $\alpha + t\varepsilon$ for some "small" error $\varepsilon \in R$ (i.e., all the coefficients of $t\varepsilon \in R$ are considerably smaller than q_i). Then the decryption procedure $\mathsf{Dec}(v_i, \mathsf{sk}) = [\langle \mathsf{sk}, v_i \rangle]_{q_i} \bmod t$ can recover the correct plaintext.

A public key is a pair $\mathsf{pk} = (a, b) \in R_{q_L}^2$, where a is uniformly sampled from R_{q_L} and $b = [as + te]_{q_L}$ is generated by sampling e from the distribution χ. A "fresh" ciphertext (i.e., an L-th level ciphertext) $v_L = \mathsf{Enc}(\alpha, \mathsf{pk}) = (c_0, c_1) \in R_{q_L}^2$ of a plaintext $\alpha \in R_t$ is given by

$$\begin{cases} c_0 = \alpha + bv + te_0, \\ c_1 = av + te_1, \end{cases}$$

where $v \in R$ is a randomly chosen polynomial with coefficients in $\{0, \pm 1\}$, and $e_0, e_1 \in R$ are chosen from χ. Since

$$\begin{aligned} c_0 - c_1 s &= \alpha + bv + te_0 - s(av + te_1) \\ &\equiv \alpha + t(ve + e_0 - se_1) \bmod q_L, \end{aligned} \tag{2}$$

the noise polynomial of v_L satisfies the requirement of correct decryption if large q_L is taken (note that every coefficient of $ve + e_0 - se_1$ is small). Other i-th level ciphertexts, computed after homomorphic operations, are described below.

2.2 Homomorphic Operations and Noise Control

The noise polynomial of a ciphertext should be small for correct decryption. In `HElib`, the size of such noise is measured by the *canonical embedding norm* (see [13, Appendix A]). Specifically, every i-th level ciphertext $(c_0, c_1) \in R_{q_i}^2$ is

represented as $((c_0, c_1), i, \nu)$, where ν is an estimate on the noise magnitude in the ciphertext. For example, according to equation (2), an estimate for a fresh ciphertext is initialized by four parameters (N, t, σ, H) in `HElib` (see [14, Subsection 3.1.4]). Given $\mathfrak{c} = ((c_0, c_1), i, \nu)$ and $\mathfrak{c}' = ((c'_0, c'_1), i', \nu')$, homomorphic operations are defined as follows:

1. *Homomorphic Addition*: If $i = i'$, we add these two ciphertext vectors and two noise estimates as

$$\mathfrak{c} + \mathfrak{c}' = (([c_0 + c'_0]_{q_i}, [c_1 + c'_1]_{q_i}), i, \nu + \nu').$$

   If $i \neq i'$, we reduce the larger one modulo the smaller of the two moduli to bring them to the same level.
2. *Homomorphic Multiplication*: If needed, we bring the two ciphertexts to the same level. When $i = i'$, we first perform the tensor product of $\mathfrak{c}$ and $\mathfrak{c}'$ as $((d_0, d_1, d_2), i, \nu\nu')$, where (d_0, d_1, d_2) is the extended ciphertext $(c_0c'_0, c_0c'_1 + c'_0c_1, c_1c'_1) \in R^3_{q_i}$. We then perform the "relinearlization/key-switching" operation [14, Subsection 3.1.4] to obtain a *canonical* ciphertext $(c''_0, c''_1) \in R^2_{q_i}$ of the same plaintext, and estimate its noise magnitude ν''. As a result, homomorphic multiplication outputs the information

$$\mathfrak{c} * \mathfrak{c}' = ((c''_0, c''_1), i, \nu'').$$

The homomorphic correctness of the BGV scheme follows the ring structure of the plaintext space R_t; For two i-th level ciphertexts $\mathfrak{c}$ and $\mathfrak{c}'$ of plaintexts α and α', we have

$$\begin{cases} \mathsf{Dec}(\mathfrak{c} + \mathfrak{c}', \mathsf{sk}) = \alpha + \alpha', \\ \mathsf{Dec}(\mathfrak{c} * \mathfrak{c}', \mathsf{sk}) = \alpha\alpha', \end{cases} \tag{3}$$

if the i-th level modulus q_i is sufficiently large. In contrast, every homomorphic operation grows the noise of a ciphertext, and it might fail to decrypt the ciphertext after a number of operations. In particular, homomorphic multiplication grows the noise much larger than addition, and the noise grows only linearly with the number of multiplications. The "modulus-switching" operation takes as input an i-th level ciphertext to output an $(i-1)$-th level ciphertext of the same plaintext, but it can reduce the noise size (see [14, Subsection 3.1.5]). In `HElib`, if one of the input ciphertexts for homomorphic multiplication have larger noises than a preset constant, the modulus-switching is performed before the multiplication in order to avoid decryption failure. Namely, noises of ciphertexts are automatically controlled in `HElib`. Then we can ignore the noise magnitude and the level of a ciphertext in using `HElib`.

2.3 Plaintext Slots of the BGV Scheme

Here we take a prime p as the plaintext modulus for slots. The N-th cyclotomic polynomial factors modulo p into k irreducible factors as $\Phi_N(x) \equiv F_1(x) \times$

$F_2(x) \times \cdots \times F_k(x) \pmod{p}$ for some $k \in \mathbb{Z}$. Every polynomial $F_i(x) \in \mathbb{F}_p[x]$ has degree $d = \phi(N)/k$. Due to the ring-isomorphism

$$R_p \simeq \mathbb{F}_p[x]/(F_1(x)) \times \cdots \times \mathbb{F}_p[x]/(F_k(x)), \tag{4}$$

any plaintext polynomial $\alpha \in R_p$ can be identified as the vector $(\alpha \bmod F_i(x))_{i=1}^k$. Conversely, since every space $\mathbb{F}_p[x]/(F_i(x))$ is isomorphic to the extension field $\mathbb{F}_{p^d}$, we can handle any vector $\mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{F}_{p^d}^k$ as a plaintext (each entry is called a "slot"), and encrypt the vector as $\mathsf{Enc}(\mathbf{a}, \mathsf{pk})$. By the isomorphism (4), for two ciphertexts $\mathfrak{c}, \mathfrak{c}'$ of vectors $(a_1, \ldots, a_k), (b_1, \ldots, b_k)$, it satisfies

$$\begin{cases} \mathsf{Dec}(\mathfrak{c} + \mathfrak{c}', \mathsf{sk}) = (a_1 + b_1, \ldots, a_k + b_k), \\ \mathsf{Dec}(\mathfrak{c} * \mathfrak{c}', \mathsf{sk}) = (a_1 b_1, \ldots, a_k b_k). \end{cases} \tag{5}$$

Namely, homomorphic addition (resp., multiplication) allows us to operate element-wise addition (resp., multiplication) over plaintext slots (cf., the homomorphic property (3) over the native plaintext space R_t). Simply speaking, it enables SIMD operation over slots (see [13] for a typical use). Furthermore, procedures for data-movement such as rotation and shift over slots are implemented in `HElib`.

2.4 Other Optimizations

We describe the following two optimizations implemented in `HElib`, which shall be used in our implementation.

1. Modulus Chain and Double-CRT Representation [14, Subsection 1.2]: For the moduli chain (1), small primes $p_0, p_1, \ldots, p_L$ are chosen so that $\Phi_N(x)$ factors modulo p_i to linear terms for all $0 \le i \le L$. The ℓ-th modulus in the chain is defined as $q_\ell = \prod_{j=0}^{\ell} p_j$ in `HElib`. For efficient arithmetic over ciphertexts, a polynomial $a(x) \in R_{q_\ell}$ (in coefficient representation) is represented as an $(\ell + 1) \times \phi(N)$ matrix $\mathsf{DoubleCRT}^\ell(a)$ (in *evaluation representation*), whose (i, j)-th entry is the evaluation of $a(x)$ at j-th root of $\Phi_N(x)$ modulo p_i. Addition and multiplication in R_{q_ℓ} is done entry-wise modulo the appropriate primes p_i. In the rest of description, we ignore the double CRT representation of polynomials lying in the ciphertext space and describe the scheme as if we were operating on polynomials directly.
2. Key Switching [14, Subsection 3.1.6]: The public key has key switching matrices that transform a ciphertext decryptable by a secret key sk' into a ciphertext decryptable by another secret key sk. This transformation is used in multiplication and data-movement over slots (in particular, this is an optimization in getting a canonical ciphertext).

3 Secure Matrix Multiplications

In this section, we review typical known methods for secure matrix multiplications over ring-LWE based homomorphic encryption (see also Figure 1 for its

image). We also extend the method of Duong et al. [9] for secure *multiple* matrix multiplications. For simplicity, in this paper, we only handle square matrices of size m with positive integer entries.

3.1 Typical Known Methods

Matrix-Vector Multiplication in `HElib` To perform secure matrix multiplications, several methods for the matrix-vector multiplication over the BGV scheme are implemented in `HElib`. According to [15, Subsection 4.3], the method to put a matrix in *diagonal order* is the best solution if the matrix is given to us in plaintext (cf., if a matrix is encrypted, expensive data movement techniques in `HElib` are required to change the representation of the matrix). Let $\mathbf{A} = (a_{ij})$ be an integer square matrix of size m, and $\mathbf{v} = (v_i)$ a column vector of length m. We represent $\mathbf{A}$ by m column vectors in diagonal order as $\mathbf{d}_1 = (a_{1,1}, a_{2,2}, \ldots, a_{m,m})$ and

$$\mathbf{d}_i = (a_{1,i}, a_{2,i+1}, \ldots, a_{m-i+1,m}, a_{m-i+2,1}, \cdots, a_{m,i-1})$$

for $2 \leq i \leq m$. Then the product $\mathbf{A} \times \mathbf{v}$ can be computed as $\sum_{i=1}^{m} \mathbf{d}_i \times (\mathbf{v} \lll i) \in \mathbb{Z}^m$, where $(\mathbf{v} \lll i)$ is the i times left-rotated vector of $\mathbf{v}$ and $\mathbf{a} \times \mathbf{b}$ is the element-wise multiplication between two vectors $\mathbf{a}$ and $\mathbf{b}$. To evaluate this procedure over the BGV scheme, we encrypt vectors $\mathbf{d}_i$ and $\mathbf{v}$ encoded in slots as $\mathfrak{c}_i = \mathsf{Enc}(\mathbf{d}_i, \mathsf{pk})$ and $\mathfrak{c} = \mathsf{Enc}(\mathbf{v}, \mathsf{pk})$ (it requires at least m slots). From the homomorphic property (5) over slots, a combination of homomorphic operations

$$\sum_{i=1}^{m} \mathfrak{c}_i * \mathsf{Rotate}_{i-1}(\mathfrak{c})$$

outputs a ciphertext of the product $\mathbf{A} \times \mathbf{v}$, where $\mathsf{Rotate}_j(\mathfrak{c})$ is the j times left-rotated ciphertext of $\mathfrak{c}$. This requires $(m-1)$ rotations and additions, and m multiplications. The hoisting technique mentioned in [17] can be used to make rotations more efficient when multiple rotations are to be performed on the same ciphertext. This matrix-vector multiplication can be applied to matrix multiplications. (Recently, Lu et al. [20] slightly modified the matrix-vector multiplication of [15, Section 4.3] for homomorphic evaluation of principal component analysis and linear regression.)

Duong et al.'s Method for Single Matrix Multiplication While the above method makes use of the homomorphic property (5) over slots, Duong et al. [9] use the property (3) over the native plaintext space R_t of the BGV scheme (they in [9] implemented their method over the BV scheme [5], the origin of the BGV scheme without optimizations).

For a 2-power integer n, we here take $N = 2n$ as the parameter defining the base ring R of the BGV scheme. Then we have $\Phi_N(x) = x^n + 1$ and hence

$$R = \mathbb{Z}[x]/(x^n + 1).$$

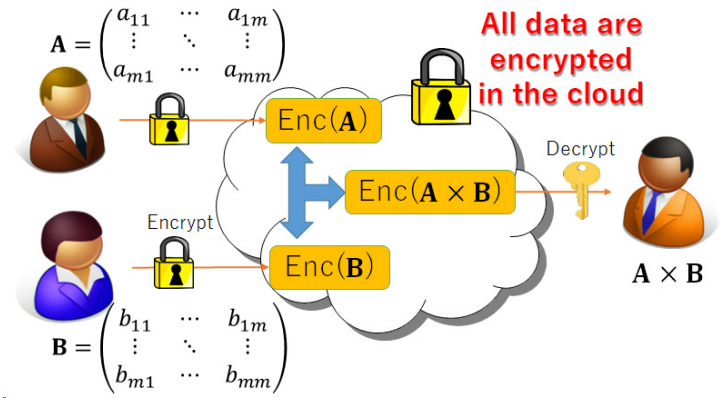

Fig. 1. An image of secure matrix multiplication $\mathbf{A} \times \mathbf{B}$ in the cloud

Let $\mathbf{A}$ and $\mathbf{B}$ be two square matrices of size m with positive integer entries. We pack each row $A_i = (a_{i,0}, \ldots, a_{i,m-1})$ and column $B_j^T = (b_{0,j}, \ldots, b_{m-1,j})$ of matrices $\mathbf{A}$ and $\mathbf{B}$, respectively, into polynomials of R as follows:

$$\begin{cases} \mathrm{pm}_2^{(1)}(A_i) := \sum_{u=0}^{m-1} a_{i,u} x^u, \\ \mathrm{pm}_2^{(2)}(B_j^T) := -\sum_{v=0}^{m} b_{v,j} x^{n-v}. \end{cases}$$

Furthermore, define the following two types of polynomial in R associated to two matrices $\mathbf{A}$ and $\mathbf{B}$:

$$\begin{cases} \mathrm{Pol}_2^{(1)}(\mathbf{A}) := \sum_{i=1}^{m} \mathrm{pm}_2^{(1)}(A_i) x^{(i-1)m}, \\ \mathrm{Pol}_2^{(2)}(\mathbf{B}) := \sum_{j=1}^{m} \mathrm{pm}_2^{(2)}(B_j^T) x^{(j-1)m^2}. \end{cases}$$

We then encrypt the two types of polynomial as

$$\mathfrak{ct}_2^{(i)}(\mathbf{A}) := \mathsf{Enc}\left(\mathrm{Pol}_2^{(i)}(\mathbf{A}), \mathsf{pk}\right) \quad \text{for } i = 1, 2.$$

Over packed ciphertexts, we have the following result [9, Theorem 7] (they modify the result for large entries in [9, Subsection 4.2], based on Yasuda et al.'s modification [31]):

Theorem 1. *Assume $n \geq m^3$. Let*

$$\mathfrak{ct}_2 = \mathfrak{ct}_2^{(1)}(\mathbf{A}) * \mathfrak{ct}_2^{(2)}(\mathbf{B})$$

and let $\mathsf{Dec}(\mathfrak{ct}_2, \mathsf{sk}) \in R_t$ *denote its decryption polynomial. If decryption is correct for* $\mathfrak{ct}_2$*, then the inner product* $\langle A_i, B_j^T \rangle$ *modulo* t *is equal to the coefficient of* $x^{(i-1)m+(j-1)m^2}$ *in* $\mathsf{Dec}(\mathfrak{ct}_2, \mathsf{sk})$ *for every* $1 \leq i, j \leq m$*.*

Proof. Here we only give a sketch of proof. Due to the homomorphic property (3) over the native plaintext space R_t, the decryption result of $\mathfrak{ct}_2$ is equal to

$$F(x) := \mathrm{Pol}_2^{(1)}(\mathbf{A}) \times \mathrm{Pol}_2^{(2)}(\mathbf{B})$$

modulo t. All the entries $\langle A_i, B_j^T \rangle$ of $\mathbf{A} \times \mathbf{B}$ are obtained from coefficients of the polynomial $F(x) \in R_t$ at certain positions. This is due to the structure of the special ring $R_t = \mathbb{F}_t[x]/(x^n + 1)$. This idea is based on Yasuda et al.'s method [30] for secure multiple inner products, but suitable arrangement of entries of a matrix is required for secure matrix multiplication to avoid overlaps. For details, see the proof of Theorem 2 below for the case of three matrices.

By Theorem 1, if we take sufficiently large t, all the entries of $\mathbf{A} \times \mathbf{B}$ can be obtained from coefficients of the decryption polynomial $\mathsf{Dec}(\mathfrak{ct}_2, \mathsf{sk}) \in R_t$. This method can pack a matrix into a single ciphertext, and it requires only one homomorphic multiplication over packed ciphertexts for a single secure matrix multiplication $\mathbf{A} \times \mathbf{B}$. (cf., the method described in Subsection 3.1 packs a row or column vector of a matrix into a single ciphertext, and it requires at least m^2 homomorphic multiplications for $\mathbf{A} \times \mathbf{B}$.) An obstacle of Duong et al.'s method is that it requires considerably large n and t for large size m and entries of matrices (see Subsection 4.1 below for this obstacle).

Remark 1. In Duong et al.'s method, types of packed polynomial are different for two matrices $\mathbf{A}$ and $\mathbf{B}$, and hence we can not change the order of matrices for multiplication. In [28], Wang et al. modified Duong et al.'s packing method so that it can pack a matrix into a single ciphertext in the same way, and it can also compute both $\mathbf{A} \times \mathbf{B}$ and $\mathbf{B} \times \mathbf{A}$ simultaneously over packed ciphertexts. However, Wang et al.'s method requires larger n for the ring $R = \mathbb{Z}[x]/(x^n + 1)$ to pack a matrix, and it would be much less efficient than Duong et al.'s method for matrices of larger size.

3.2 Generalization of Duong et al.'s Method for Multiple Matrix Multiplications

The method by Duong et al. [9] is only for a single secure multiplication. Here we generalize their method to *multiple* matrix multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$. Throughout this subsection, we let A_i (resp., A_i^T) denote the i^{th} row (resp., column) of a matrix $\mathbf{A}$. We use the notation "pm_ℓ" for packing a row (or a column) of matrix, where ℓ stands for the number of matrices being multiplied.

Case of $\ell = 3$ We begin to consider the case of three matrices. Let $\mathbf{A}$ be a square matrix of size m whose entries are positive integers. As in the previous

subsection, we give two types of polynomial in $R = \mathbb{Z}[x]/(x^n + 1)$ for each row $A_i = (a_{i,0}, \ldots, a_{i,m-1})$ of $\mathbf{A}$ as follows:

$$\begin{cases} \mathrm{pm}_3^{(1)}(A_i) := \sum_{u=0}^{m-1} a_{i,u} x^u, \\ \mathrm{pm}_3^{(2)}(A_i) := -\sum_{u=0}^{m-1} a_{i,u} x^{n-um^2-m+1}, \end{cases}$$

Now we define three types of polynomial in R associated with three matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ as follows:

$$\begin{cases} \mathrm{Pol}_3^{(1)}(\mathbf{A}) := \sum_{i=1}^{m} \mathrm{pm}_3^{(1)}(A_i)\, x^{(i-1)m}, \\ \mathrm{Pol}_3^{(2)}(\mathbf{B}) := \sum_{j=1}^{m} \mathrm{pm}_3^{(1)}\left(\overline{B}_j^T\right) x^{(j-1)m^2}, \\ \mathrm{Pol}_3^{(3)}(\mathbf{C}) := \sum_{k=1}^{m} \mathrm{pm}_3^{(2)}\left(C_k^T\right) x^{(k-1)m^3}, \end{cases}$$

where $B_j^T = (b_{0,j}, \ldots, b_{m-1,j})$ and C_k^T are the j^{th} and the k^{th} columns of $\mathbf{B}$ and $\mathbf{C}$, respectively, and set

$$\overline{B}_j^T = (b_{m-1,j}, \ldots, b_{0,j})$$

(i.e., flipping the column B_j^T). Now we define three types of packed ciphertext for a matrix $\mathbf{A}$ to be

$$\mathfrak{ct}_3^{(i)}(\mathbf{A}) := \mathsf{Enc}\left(\mathrm{Pol}_3^{(i)}(\mathbf{A}), \mathsf{pk}\right) \quad \text{for } i = 1, 2, 3.$$

Then we have a similar result to Theorem 1 for the case of three matrices as follows:

Theorem 2. *Assume $n \geq m^4$. Let*

$$\mathfrak{ct}_3 = \mathfrak{ct}_3^{(1)}(\mathbf{A}) * \mathfrak{ct}_3^{(2)}(\mathbf{B}) * \mathfrak{ct}_3^{(3)}(\mathbf{C}),$$

and let $\mathsf{Dec}(\mathfrak{ct}_3, \mathsf{sk}) \in R_t$ denote its decryption polynomial. If decryption is correct for $\mathfrak{ct}_3$, then the $(i,k)^{\text{th}}$ entry of $\mathbf{A} \times \mathbf{B} \times \mathbf{C}$ modulo t is equal to the coefficient of $x^{(i-1)m+(k-1)m^3}$ in $\mathsf{Dec}(\mathfrak{ct}_3, \mathsf{sk})$ for every $1 \leq i, k \leq m$.

Proof. The idea is the same as the proof of Theorem 1. It follows from our construction that

$$\begin{aligned} &\mathrm{pm}_3^{(1)}(A_i) \times \mathrm{pm}_3^{(1)}\left(\overline{B}_j^T\right) \\ &= \sum_{u=1}^{m} \sum_{v=1}^{m} a_{iu} b_{m-v+1,j} x^{u+v-2} \\ &= \left\langle A_i, B_j^T \right\rangle x^{m-1} + \text{other terms of degree } (u+v-2) \end{aligned}$$

with $u \neq m - v + 1$ and $u, v \in \{0, \ldots, m-1\}$. As seen from the above equation, the coefficient of the monomial x^{m-1} in $\mathrm{pm}_3^{(1)}(A_i) \times \mathrm{pm}_3^{(1)}\left(\overline{B}_j^T\right) \in R$ gives the inner product $\left\langle A_i, B_j^T \right\rangle$. Furthermore we have

$$\begin{aligned}
&\mathrm{pm}_3^{(1)}(A_i) \times \mathrm{Pol}_3^{(2)}(\mathbf{B}) \\
=&\sum_{j=1}^{m} \left\langle A_i, B_j^T \right\rangle x^{(j-1)m^2+m-1} + \text{other terms of degree } (u+v-2+(j-1)m^2).
\end{aligned}$$

Furthermore, since $x^n = -1$ holds over R, we obtain

$$\begin{aligned}
&\mathrm{pm}_3^{(1)}(A_i) \times \mathrm{Pol}_3^{(2)}(\mathbf{B}) \times \mathrm{pm}_3^{(2)}\left(C_k^T\right) \\
=&\sum_{j=1}^{m} \left\langle A_i, B_j^T \right\rangle x^{(j-1)m^2+m-1} \times \mathrm{pm}_3^{(2)}\left(C_k^T\right) \\
&\qquad + \text{other terms of degree } (n+u+v-m-1+(j-1)m^2-(w-1)m^2) \\
=&-\sum_{j=1}^{m}\sum_{w=1}^{m} \left\langle A_i, B_j^T \right\rangle c_{k,w} x^{n+(j-1)m^2-(w-1)m^2} \\
&\qquad\qquad + \text{other terms of degree } (n+u+v-m-1+(j-w)m^2) \\
=&\left\langle V_i, C_k^T \right\rangle + \text{other terms of degree } (n+u+v-m-1+(j-w)m^2)
\end{aligned}$$

with $w \neq j$ for $w, j \in \{0, \ldots, m-1\}$, where we set $V_i = \left(\left\langle A_i, B_j^T \right\rangle\right)_{j=1}^{m}$ for $1 \leq i \leq m$. If decryption is correct for the ciphertext $\mathfrak{ct}_3$, we have

$$\begin{aligned}
\mathsf{Dec}(\mathfrak{ct}_3, \mathsf{sk}) =&\mathrm{Pol}_3^{(1)}(\mathbf{A}) \times \mathrm{Pol}_3^{(2)}(\mathbf{B}) \times \mathrm{Pol}_3^{(3)}(\mathbf{C}) \\
=&\sum_{i=1}^{m}\sum_{k=1}^{m} \mathrm{pm}_3^{(1)}(A_i) \times \mathrm{Pol}_{m,3}^{(2)}(\mathbf{B}) \times \mathrm{pm}_3^{(2)}\left(C_k^T\right) x^{(i-1)m+(k-1)m^3}.
\end{aligned}$$

Furthermore, for fixed indices i and k, we have

$$\begin{aligned}
&\mathrm{pm}_3^{(1)}(A_i) \times \mathrm{Pol}_3^{(2)}(\mathbf{B}) \times \mathrm{pm}_3^{(2)}\left(C_k^T\right) x^{(i-1)m+(k-1)m^3} \\
=&\left\langle V_i, C_k^T \right\rangle x^{(i-1)m+(k-1)m^3} \\
&+ \text{other terms of degree } (n+u+v-m-1+(j-w)m^2+(i-1)m+(k-1)m^3
\end{aligned}$$

with $u+v \neq m+1$ and $w \neq j$ for $u, v, j, w \in \{0, \ldots, m-1\}$. The exponent of x is modulo n over R, and therefore $n+u+v-m-1+(j-w)m^2+(h-1)m+(\ell-1)m^3$ is never equal to $(i-1)m+(k-1)m^3$ for $u+v \neq m+1, w \neq j$ and $j, w, h, \ell, i, k \in \{1, \ldots, m\}$. This implies that the term of degree $(i-1)m+(k-1)m^3$ in

$$\mathrm{Pol}_3^{(1)}(\mathbf{A}) \times \mathrm{Pol}_3^{(2)}(\mathbf{B}) \times \mathrm{Pol}_3^{(3)}(\mathbf{C})$$

is exactly the term of degree $(i-1)m+(k-1)m^3$ in

$$\mathrm{pm}_3^{(1)}(A_i) \times \mathrm{Pol}_3^{(2)}(\mathbf{B}) \times \mathrm{pm}_3^{(2)}\left(C_k^T\right) x^{(i-1)m+(k-1)m^3}.$$

Hence the inner product $\langle V_i, C_k^T \rangle$, the $(i,k)^{th}$ entry of $\mathbf{A} \times \mathbf{B} \times \mathbf{C}$, can be recovered from the coefficient of $x^{(i-1)m+(k-1)m^3}$ in the decryption polynomial $\mathsf{Dec}(\mathfrak{ct}_3, \mathsf{sk}) \in R_t$.

Case of $\ell \geq 4$ Let $\mathbf{A}$ be a square matrix of size m. For more than three matrices, we define $(\ell-1)$ types of polynomial in R for each row $A_i = (a_{i,0}, \ldots, a_{i,m-1})$ of $\mathbf{A}$ as follows:

$$\begin{cases} \mathrm{pm}_\ell^{(1)}(A_i) := \sum_{u=0}^{m-1} a_{i,u} x^u, \\ \mathrm{pm}_\ell^{(2)}(A_i) := -\sum_{u=0}^{m-1} a_{i,u} x^{n-(\alpha+\beta+m-1)}, \\ \mathrm{pm}_\ell^{(w)}(A_i) := \sum_{u=0}^{m-1} a_{i,u} x^{um^{w-1}} \text{ for } 3 \leq w \leq \ell-1, \end{cases}$$

where $\alpha = um^{\ell-1}$ and $\beta = \sum_{s=2}^{\ell-2}(m-1)m^s$. Furthermore, we define ℓ types of polynomial in R associated to matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ as follows:

$$\begin{cases} \mathrm{Pol}_\ell^{(1)}(\mathbf{A}_1) := \sum_{i=1}^{m} \mathrm{pm}_\ell^{(1)}(A_{1,i})\, x^{(i-1)m}, \\ \mathrm{Pol}_\ell^{(2)}(\mathbf{A}_2) := \sum_{j=1}^{m} \mathrm{pm}_\ell^{(1)}\left(\overline{A}_{2,j}^T\right) x^{(j-1)m^2}, \\ \mathrm{Pol}_\ell^{(h)}(\mathbf{A}_h) := \sum_{k=1}^{m} \mathrm{pm}_\ell^{(h)}\left(\overline{A}_{h,k}^T\right) x^{(k-1)m^h} \ (3 \leq h \leq \ell-1), \\ \mathrm{Pol}_\ell^{(\ell)}(\mathbf{A}_\ell) := \sum_{w=1}^{m} \mathrm{pm}_\ell^{(2)}\left(A_{\ell,w}^T\right) x^{(w-1)m^\ell}, \end{cases}$$

where $A_{h,j}$ and $A_{h,j}^T = \left(a_{0,j}^{(h)}, \ldots, a_{m-1,j}^{(h)}\right)$ are the row and column of $\mathbf{A}_h$ respectively, and $\overline{A}_{h,j}^T = \left(a_{m-1,j}^{(h)}, \ldots, a_{0,j}^{(h)}\right)$ (i.e., flipping the column $A_{h,j}^T$) for $3 \leq h \leq \ell-1$. Define ℓ types of packed ciphertext for a given matrix $\mathbf{A}$ to be

$$\mathfrak{ct}_\ell^{(h)}(\mathbf{A}) := \mathsf{Enc}\left(\mathrm{Pol}_\ell^{(h)}(\mathbf{A}), \mathsf{pk}\right) \quad \text{for } 1 \leq h \leq \ell.$$

Similar to Theorem 2, we obtain the following result for secure matrix multiplications over packed ciphertexts:

Theorem 3. *Assume $n \geq m^{\ell+1}$. Let*

$$\mathfrak{ct}_\ell = \prod_{h=1}^{\ell} \mathfrak{ct}_\ell^{(h)}(\mathbf{A}_h)$$

and let $\mathsf{Dec}(\mathtt{ct}_\ell, \mathsf{sk}) \in R_t$ *denote its decryption polynomial. If decryption is correct for* $\mathtt{ct}_\ell$*, then the* $(i,j)^{\text{th}}$ *entry of* $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ *modulo* t *is equal to the coefficient of* $x^{(i-1)m+(j-1)m^\ell}$ *in* $\mathsf{Dec}(\mathtt{ct}_\ell, \mathsf{sk})$ *for every* $1 \leq i, j \leq m$*.*

4 Implementation Results

In this section, we show implementation results for our secure multiple matrix multiplications (described in Subsection 3.2) over `HElib` [16]. Specifically, we show performance of secure matrix multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$, where every $\mathbf{A}_i$ is a square matrix of size m with positive integer entries of p-bit for $\ell = 2, 3, 4$, $m = 16, 32$ and $p = 16, 32$. Note that the case $\ell = 2$ stands for Duong et al.'s method [9], presented in Subsection 3.1.

4.1 Some Modifications

As described in the paragraph before Remark 1, Duong et al.'s and our packing methods require very large n (degree of $\Phi_N(x)$ for $N = 2n$) and t (plaintext modulus) for matrices with large size and entries. (This problem is not specific to our method, and in particular, such large t might be required in any technique.) Such setting makes the BGV scheme very slow. Here we give two modifications to relax the problem.

Block-Matrix Method Let $\mathbf{A}$ be a square matrix of size m. We take a block size M satisfying $m = bM$ for some $b \in \mathbb{Z}$. Consider $\mathbf{A}$ as a matrix with b^2 sub-matrices $\mathbf{A}_{ij}$ with size $M \times M$ for $1 \leq i, j \leq b$. In this method, we pack each sub-matrix $\mathbf{A}_{ij}$ into a single polynomial by our packing method, and encrypt the polynomial. This enables us to take small n for packing every $M \times M$ sub-matrix $\mathbf{A}_{ij}$, instead of the whole matrix $\mathbf{A}$ (this requires $n \geq M^{\ell+1}$, instead of $n \geq m^{\ell+1}$). On the other hand, this method requires more homomorphic operations. For example, it requires b^3 homomorphic multiplications and $b^2(b-1)$ homomorphic additions for secure matrix multiplication between two matrices.

CRT Method for Plaintext Modulus We can split the plaintext modulus t into e small different primes t_i as

$$t = \prod_{i=1}^{e} t_i$$

for some $e \in \mathbb{Z}$. Given a message modulo t, we encrypt the plaintext modulo every t_i. After decryption modulo every t_i, we can recover the original message by the CRT method. This enables us to use $\mathbb{F}_{t_i}[x]/(x^n + 1)$ as the plaintext space for every t_i. On the other hand, it requires e ciphertexts for encrypting a message. (This problem can be alleviated through multi-threading. Moreover, smaller plaintext moduli require less number of moduli for leveled ciphertext spaces.)

4.2 Selection of Parameters

Here we describe how to select suitable parameters of the BGV scheme for our secure matrix multiplications. In our experiments, we use our CRT method with $e = 4$, suitable for running our programs over 4 threads (it requires to set different primes t_i for $1 \leq i \leq 4$). In using `HElib`, we basically need to select the following three parameters (see [14, Section 5] for usage):

- n: 2-power integer defining $R = \mathbb{Z}[x]/(x^n + 1)$
- t_i: prime moduli of plaintext spaces for $1 \leq i \leq 4$
- L: number of moduli for leveled ciphertext spaces

For other parameters in `HElib`, we use default parameters such as $\sigma = 3.2$ (standard deviation parameter for discrete Gaussian distribution) and $H = 64$ (Hamming weight of secret keys). We also set $k = 128$ as the security parameter (actually, our chosen parameters have much more than 128-bit security level from the estimate of `HElib` since our method requires large n). In our experiments, we use our block-matrix method with block size $M = 16$ (resp., $M = 8$) for $\ell = 2$ (resp., $\ell = 3, 4$). Then we select a 2-power integer n satisfying $n \geq M^{\ell+1}$, suitable for both security and efficiency. Since every coefficient of $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ in our block-matrix method is at most $M^{\ell-1}2^{\ell p}$, it needs to take $t > M^{\ell-1}2^{\ell p}$ for obtaining the correct decryption result, where p is the maximum bit size of entries of $\mathbf{A}_i$ and t is the plaintext modulus without our CRT method. Then we take 4 different small primes for t_i's satisfying

$$t_i > \left(M^{\ell-1}2^{\ell p}\right)^{1/4}.$$

In contrast, choice of the parameter L depends on the number ℓ of *successive* homomorphic multiplications and the size of plaintext space t_i to accept any noises in evaluation. For our experiments, we select around $2\ell \sim 3\ell$ for the parameter L. (See Table 1 below for our chosen parameters.)

4.3 Implementation and Performance Results

Implementation We implemented our secure matrix multiplications. Our experiments ran on an Intel(R)Xeon(R) CPU E5-46170@2.90GHz, using `HElib` [16] as a library. In our experiments, we make use of multi-threaded implementation (4 threads), compatible with our CRT-method. (cf., Our pre-experiments show that computation over 4 threads is about 2.5 times faster than a single thread.) In particular, we use the half-primes optimization in making the moduli chain (1) for efficiency (see [13, Section 3]).

Performance Results In Table 1, we show our chosen parameters of the BGV scheme and running time for secure matrix multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$, where every square matrix $\mathbf{A}_i$ has size m and p-bit positive integer entries. In the following, we compare with known performance results for each of $\ell = 2, 3, 4$:

Table 1. Performance of our secure matrix multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ over `HElib`, where every square matrix $\mathbf{A}_i$ has size m and p-bit integer entries (we represent $\mathbf{A}_i$ as $M \times M$ sub-matrices in our block-matrix method, and split the plaintext modulus into 4 primes $t_1, \ldots, t_4$ in our CRT method)

Setting of matrices				**Parameters of the scheme**			**Performance (Seconds)**			
ℓ	m	M	p	n	$\lfloor \log_2(t_i) \rfloor$	L	Encryption	Matrix Mul.	Decryption	Total Time
2	16	16	16	8192	9	5	0.0588	0.0379	0.0171	0.1140
			32	8192	18	5	0.0582	0.0462	0.0174	0.1219
	32	16	16	8192	10	5	0.1495	0.3103	0.0413	0.5012
			32	8192	18	5	0.3202	0.3436	0.0378	0.7017
3	16	8	16	16384	14	7	0.4944	1.1257	0.0970	1.7173
			32	16384	26	9	0.6183	1.2187	0.1027	1.9399
	32	8	16	16384	15	7	1.2954	9.2480	0.2116	10.7552
			32	16384	27	9	1.3680	10.3408	0.2118	11.9207
4	16	8	16	32768	19	9	1.4408	4.3138	0.2001	5.9547
			32	32768	35	11	1.5811	4.5999	0.2456	6.4266
	32	8	16	32768	20	9	3.7801	30.6657	0.4299	34.8759
			32	32768	36	11	3.7211	32.0827	0.5195	36.3234

1. For the case $\ell = 2$, our method is the same as Duong et al.'s method [9]. For two square matrices of size $m = 16$ with $p = 10$-bit entries, Duong et al.'s packing method (the modified version for large entries) in [9, Subsection 4.2] needs $n = 131072$ by [9, Theorem 10], and it took about 7.27 seconds from [9, Table 2]. In contrast, Table 1 shows that it took only 0.1140 seconds for $m = 16$ and $p = 16$. This is mainly due to our modifications since $n = 8192$ is sufficient in our block-matrix method.
2. For the case $\ell = 3$, performance results are given in our conference paper [22]. From [22, Table 2], $n = 65536$ is set without our CRT-method and it took 45.8430 (resp., 58.7190) seconds for $m = 32$ and $p = 16$ (resp., $p = 32$) over a single thread of Intel Core i7-4790 CPU with 3.60 GHz, using `PARI` library [27] (version 2.9.2) in `C` programs for implementing the BV scheme [5]. In this full version paper, we use $M = 8$ as the size of sub-matrices (cf., $M = 16$ is used in [22]), and hence smaller n is sufficient. From Table 1, our method took 10.7552 (resp., 11.9207) seconds for $m = 32$ and $p = 16$ (resp., $p = 32$). This is about 4 or 5 times faster than [22, Table 2], mainly due to smaller n and use of 4 threads over `HElib`. (Arithmetic in `HElib` is faster than `PARI` since it is optimized for the BGV scheme.) In particular, our implementation requires less time of both decryption and circuit because of relinearlization. It also requires less ciphertext modulus size due to modulus switching.
3. For the case $\ell = 4$, we use $M = 8$ and $n = 32768$, and it took 34.8759 (resp., 36.3234) seconds for $m = 32$ and $p = 16$ (resp., $p = 32$). (cf., The time was around 645 seconds in our `PARI` implementation for the BV scheme. This shows that relinearlization and modulus switching have a greater impact for large ℓ.) As seen from Table 1, this is about 3 or 4 times slower than the case $\ell = 3$. However, for larger ℓ, the performance might be considerably slower

since huge n is required in our method. Actually, big jumping exponents are used in our packed polynomials for large ℓ (see Subsection 3.2). See also Subsection 4.4 below for advantages and limitations of our method.

4.4 Comparison in Matrix-Vector Multiplications

As described in Subsection 3.1, the diagonal-based matrix-vector multiplication is implemented in `HElib`. For $\ell \geq 2$ and $m \geq 1$, let $\mathbf{A}_1, \cdots, \mathbf{A}_{\ell-1}$ be $\ell - 1$ square matrices of size m and $\mathbf{v}$ a column vector of length m. For a 2-power integer n, our packing method over $R = \mathbb{Z}[x]/(x^n + 1)$ enables us to perform secure matrix-vector multiplications as follows (the following result holds for $\ell \geq 4$, but a similar result for $\ell = 2, 3$ can be obtained from Subsections 3.1 and 3.2):

Theorem 4. *Let $\ell \geq 4$ and assume $n \geq m^\ell$. Let*

$$\mathfrak{ct} = \mathfrak{ct}^{(\ell)}(\mathbf{v}) \times \prod_{h=1}^{\ell-1} \mathfrak{ct}_\ell^{(h)}\left(\mathbf{A}_h\right),$$

where $\mathfrak{ct}^{(\ell)}(\mathbf{v}) = \mathsf{Enc}\left(\mathrm{pm}_\ell^{(2)}(\mathbf{v}), \mathsf{pk}\right)$ is a packed ciphertext of $\mathbf{v}$ (see also Subsection 3.2 for other packed ciphertexts). Let $\mathsf{Dec}(\mathfrak{ct}, \mathsf{sk}) \in R_t$ denote its decryption polynomial. If decryption is correct for $\mathfrak{ct}$, then the i^{th} entry of the column vector $\mathbf{A}_1 \times \cdots \times \mathbf{A}_{\ell-1} \times \mathbf{v}$ modulo t is equal to the coefficient of $x^{(i-1)m}$ in $\mathsf{Dec}(\mathfrak{ct}, \mathsf{sk})$.

Performance Comparison In Table 2, we compare the performance of our method with the diagonal-based method in `HElib` for secure matrix-vector multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_{\ell-1} \times \mathbf{v}$ for $\ell = 2, 3$ and 4, where every square matrix $\mathbf{A}_i$ and column vector $\mathbf{v}$ have size m and p-bit integer entries. We performed experiments over 4 threads with our CRT-method as in Table 1. We also used the hoisting optimization mentioned in [17] to further speed up the diagonal-based method (this optimization makes the performance about 1.3 faster in our experiments). In the following, we describe how to select parameters of the BGV schemes in our experiments for both our method and the diagonal-based method:

1. For our method, we selected parameters of the BGV scheme in the same way as in Subsection 4.2. For secure matrix-vector multiplications, we set larger M as the size of every sub-matrix than Table 1, but we used slightly smaller n for the base ring $R = \mathbb{Z}[x]/(x^n + 1)$. Since noise estimates in `HElib` are suitable for plaintext slots, our chosen parameters might be not optimal.
2. For the diagonal-based method, we used the same split plaintext moduli $t_1, \cdots, t_4$ as in our method. To define the base ring $R = \mathbb{Z}[x]/(\Phi_N(x))$ of the BGV scheme, we selected a prime of form $N = a \cdot m + 1$ with $\gcd(a, m) = 1$ for some $a \in \mathbb{Z}$ to obtain m slots in the BGV scheme (cf., our method requires $N = 2n$ for a 2-power integer n). This form of N has been chosen to get a one-dimensional encrypted array with a *good dimension* of order m for

Table 2. Performance comparison of our method (Theorem 4) with the diagonal-based method (Subsection 3.1) for secure matrix-vector multiplications $\mathbf{A}_1 \times \cdots \mathbf{A}_{\ell-1} \times \mathbf{v}$, where every square matrix $\mathbf{A}_i$ and column vector $\mathbf{v}$ have size m and p-bit integer entries. We represent $\mathbf{A}_i$ as $M \times M$ sub-matrices in our block-matrix method and vector $\mathbf{v}$ is represented as M size sub-vectors, and split the plaintext modulus into 4 primes $t_1, \cdots, t_4$ in our CRT method. The performance of our method is shown in bold face (our method uses $\mathbb{Z}[x]/(\Phi_N(x))$ for $N = 2n$ with 2-power integer n).

Setting of matrices				**Parameters of the scheme**			**Performance (Seconds)**			
ℓ	m	M	p	$\lfloor \log_2(t_i) \rfloor$	N	L	Encryption	Circuit	Decryption	Total Time
2	16	16	16	10	6553	5	0.4033	0.6496	0.0236	1.0765
					$2 \cdot 8192$	**5**	**0.0217**	**0.0367**	**0.0080**	**0.0664**
			32	18	6737	5	0.4792	0.7117	0.0397	1.02306
					$2 \cdot 8192$	**5**	**0.0237**	**0.0286**	**0.0072**	**0.0595**
	32	32	16	10	6689	5	0.8514	1.3918	0.0397	1.2306
					$2 \cdot 8192$	**5**	**0.0489**	**0.0605**	**0.0118**	**0.1212**
			32	18	7457	5	0.9925	1.5331	0.1539	2.6795
					$2 \cdot 8192$	**5**	**0.0170**	**0.0201**	**0.0051**	**0.0422**
3	16	16	16	15	8753	7	1.7826	2.9803	0.0529	4.8158
					$2 \cdot 8192$	**7**	**0.0284**	**0.0645**	**0.0051**	**0.0980**
			32	28	13649	9	2.5603	4.1327	0.2313	6.9243
					$2 \cdot 16384$	**9**	**0.0925**	**0.2523**	**0.0291**	**0.3739**
	32	32	16	15	8929	7	4.5275	5.9489	0.2178	10.6942
		16			**$2 \cdot 8192$**	**7**	**0.1408**	**0.3099**	**0.0122**	**0.4629**
		32	32	28	13217	11	6.1762	9.2014	0.1439	15.5215
		16			**$2 \cdot 16384$**	**9**	**0.3156**	**0.8628**	**0.0332**	**1.2116**
4	16	16	16	20	13649	11	4.2130	7.4059	0.0946	11.7135
		8			**$2 \cdot 16384$**	**9**	**0.4277**	**1.3914**	**0.0290**	**1.8481**
		16	32	36	20753	15	10.0600	17.5948	0.5150	28.1698
		8			**$2 \cdot 16384$**	**11**	**0.4508**	**1.5136**	**0.0272**	**1.9916**
	32	32	16	20	13217	11	8.5144	13.9225	0.1439	22.5808
		8			**$2 \cdot 16384$**	**9**	**1.9307**	**8.3040**	**0.0582**	**10.2929**
		32	32	36	20897	17	23.4615	38.7671	0.3008	62.5294
		8			**$2 \cdot 16384$**	**11**	**1.6782**	**9.0052**	**0.0633**	**10.7467**

efficient and true rotations over slots (see [15, Section 5] for good and bad dimensions). The parameter L has been chosen through experimentation, but our chosen L for the diagonal-based method is equal to or slightly larger than our method for correct decryption (around $2\ell \sim 3\ell$ was chosen for L as in our method). In practice, the diagonal-based method works efficiently when ciphertexts are multiplied one after other, and hence a larger L is required. For $\ell = 4$, the diagonal-based method takes 3 successive multiplications while ours takes 2 successive multiplications.

Remark 2. For $\ell = 4$, $m = 32$ and $p = 32$ in Table 2, our method required 3.5 GB (0.6 GB excluding context) for secure matrix-vector multiplications. (cf., it required 10.6 GB (1.2 GB excluding context) for secure matrix multiplications with same parameters.) In contrast, the diagonal-based matrix-vector multiplications required 2.8 GB (2 GB excluding context). We observed that the total

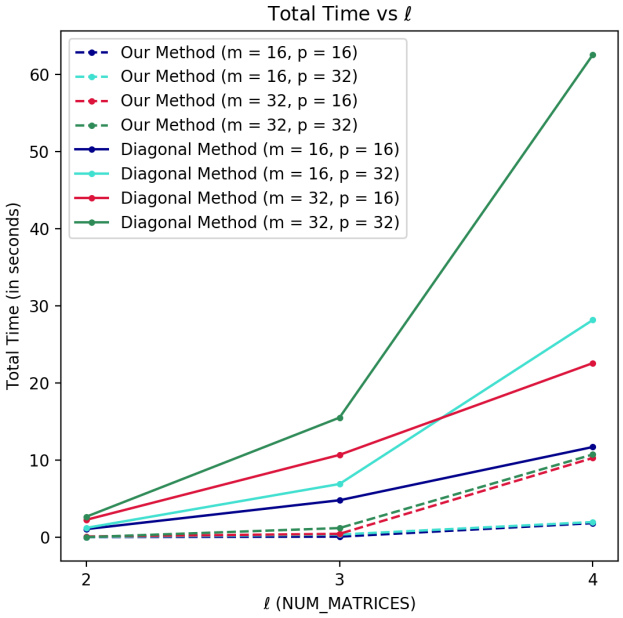

Fig. 2. Plot showing comparison of the results between our method and the diagonal-based method for secure matrix-vector multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_{\ell-1} \times \mathbf{v}$. Dashed lines and solid lines are used to represent the total time of our method and the diagonal-based method, respectively (the data are given from Table 2).

memory usage of the diagonal-based method is less than our method, but our method required less memory once the context got initialized. The context generates some components consuming a lot of memory, which are not being used by our method. Hence for a better instantiation of the scheme, without the unnecessary components of context, our method will require less memory than the diagonal-based method.

Advantages and Limitations of Our Method In Figure 2, we summarize comparison results of the total time of both our method and the diagonal-based method for secure matrix-vector multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_{\ell-1} \times \mathbf{v}$ from Table 2. In the following, we shall discuss advantages and limitations of our method (the same discussion holds for secure matrix multiplications):

Advantages As seen from Table 2 and Figure 2, our method is much faster than the diagonal-based method for $\ell = 2, 3$ and 4. This is mainly due to that as described in Section 3, our method basically requires $(\ell - 1)$ homomorphic multiplications over every thread while the diagonal-based method requires at least

$m(\ell-1)$ homomorphic multiplications and (expensive) rotations, where m is the size of every square matrix $\mathbf{A}_i$. More precisely, our method takes $O(\log \ell)$ successive multiplications while the diagonal-based method takes $O(\ell)$ successive multiplications. Hence a considerably large L is required for the diagonal-based method, as a result, a large N is required for security considerations. This is why for the slopes in Figure 2 for the diagonal-based method is much larger than ours for larger ℓ. We also see from Table 2 that encryption time of our method is considerably faster than the diagonal-based method (expensive encoding encryption in the diagonal-based method also makes the performance slow). This is due to that our method can pack a matrix (or a vector) into a single ciphertext, but the diagonal-based method can only pack a row or column vector into slots.

Limitations For large ℓ such as $\ell \geq 8$, we predict that our method would be slower than the diagonal-based method. This is due to that huge $N = 2n$ with 2-power n is required in our method while flexible primes N can be chosen in the diagonal-based method. Furthermore, since our packing method is different for each matrix $\mathbf{A}_i$, our method does not allow to operate packed ciphertexts flexibly. For example, we can not change the order of matrices for multiplications in our method as described in Remark 1. On the other hand, since the diagonal-based method can pack a row or column vector of a matrix in the same way, it enables to operate packed ciphertexts more flexibly over `HElib` with high-level procedures for data-movement over slots (see [15]).

5 Conclusion

We generalized Duong et al.'s packing method [9] for secure *multiple* matrix multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ with integer entries over the BGV scheme. Our method can pack a matrix into a single ciphertext while the diagonal-based method implemented in `HElib` can only pack a row or column vector. Furthermore, our method enables us to perform secure matrix multiplications by only a few homomorphic multiplications over packed ciphertexts. In particular, our method makes use of the homomorphic property over the native plaintext space of the BGV scheme while the diagonal-based method uses plaintext slots, suitable for SIMD. Our experimental results showed that our method is much faster than the diagonal-based method implemented in `HElib` for small ℓ such as $\ell = 2, 3$ and 4 (our method might be slower for large $\ell \geq 8$). Compared to the diagonal-based method, ours is not flexible for operating matrices in encrypted format, but it requires much less memory. Therefore we hope that our method would be useful in evaluating a *fixed* circuit including matrix multiplications.

Acknowledgment

This work was supported by JST CREST Grant Number JPMJCR14D6, Japan. This work was also supported by JSPS KAKENHI Grant Number 16H02830.

References

1. Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J Wu. Private database queries using somewhat homomorphic encryption. In *Applied Cryptography and Network Security–ACNS 2013*, volume 7954 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2013.
2. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography–TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.
3. Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding–IMACC 2013*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2013.
4. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT) - Special issue on innovations in theoretical computer science 2012 - Part II*, 6(3):13, 2014.
5. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Advances in Cryptology–CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.
6. Jung Hee Cheon, Miran Kim, and Myungsun Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Transactions on Information Forensics and Security*, 11(1):188–199, 2016.
7. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advacens in Cryptology–ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2016.
8. Ana Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? In *Topics in Cryptology–CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 325–340. Springer, 2016.
9. Dung Hoang Duong, Pradeep Kumar Mishra, and Masaya Yasuda. Efficient secure matrix multiplication over LWE-based homomorphic encryption. *Tatra Mountains Mathematical Publications*, 67(1), 2016.
10. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive, Report 2012/144*, 2012.
11. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Symposium on Theory of Computing–STOC 2009*, pages 169–178. ACM, 2009.
12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.
13. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.
14. Shai Halevi and Victor Shoup. *Design and implementation of a homomorphic-encryption library*, 2013.
15. Shai Halevi and Victor Shoup. Algorithms in HElib. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 554–571, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
16. Shai Halevi and Victor Shoup. Helib `https://github.com/shaih/HElib`, 2014.

17. Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in HElib. *Cryptology ePrint Archive, Report 2018/244*, 2018.
18. Tancrede Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In *Progress in Cryptology–AFRICACRYPT 2014*, volume 8469 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2014.
19. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing–STOC 2012*, pages 1219–1234. ACM, 2012.
20. Wenjie Lu, Shohei Kawasaki, and Jun Sakuma. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data (this is the full version of the conference paper presented at NDSS 2017). *IACR Cryptology ePrint Archive, Report 2016/1163*, 2016.
21. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):43, 2013.
22. Pradeep Kumar Mishra, Dung Hoang Duong, and Masaya Yasuda. Enhancement for secure multiple matrix multiplications over ring-LWE homomorphic encryption. In *International Conference on Information Security Practice and Experience–ISPEC 2017*, volume 10701 of *Lecture Notes in Computer Science*, pages 320–330. Springer, 2017.
23. Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop–CCSW 2011*, pages 113–124. ACM, 2011.
24. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology–EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, 1999.
25. Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
26. Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.
27. The PARI Group, Bordeaux. Pari/gp `https://pari.math.u-bordeaux.fr/download.html`.
28. Lihua Wang, Yoshinori Aono, and Le Trieu Phong. A new secure matrix multiplication from ring-LWE. In *International Conference on Cryptology and Network Security–CANS 2017*. to be published.
29. Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure pattern matching using somewhat homomorphic encryption. In *Proceedings of the 2013 ACM workshop on Cloud computing security workshop–CCSW 2013*, pages 65–76. ACM, 2013.
30. Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. New packing method in somewhat homomorphic encryption and its applications. *Security and Communication Networks*, 8(13):2194–2213, 2015.
31. Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure statistical analysis using RLWE-based homomorphic encryption. In *Australasian Conference on Information Security and Privacy–ACISP 2015*, volume 9144 of *Lecture Notes in Computer Science*, pages 471–487. Springer, 2015.