Faster Secure Matrix Computation over Homomorphic Encryption

プラディープ, クマー, ミシラ

Faster Secure Matrix Computations over Homomorphic Encryption

Pradeep Kumar Mishra

Supervisor: Prof. Masaya Yasuda

Department of Mathematics

Kyushu University

This dissertation is submitted for the degree of

Doctor of Philosophy

December 2018

I would like to dedicate this thesis to my loving parents, Mr. Kripa Shankar Mishra and Ms. Heera Devi for their love, prayers, and encouragement.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Pradeep Kumar Mishra

December 2018

Acknowledgements

I would like to thank:

My parents, my family, for supporting me at all stages of my education and their unconditional love.

My Supervisor, Prof. Masaya Yasuda, for all the support and encouragement provided to me during my work under his supervision. It would not have been possible to finish my research without his invaluable help of constructive comments and suggestions.

My Co-Supervisor, Prof. Duong Hoang Dung and Prof. Tsuyoshi Takagi, for his continuous support and encouragement.

The Japan International Cooperation Agency (JICA), for funding me with a scholarship.

My labmates from Yasuda-lab and Takagi-lab for the endless support.

My friends, Mr. Nagarjuna Asam and Mr. Deevashwer Rathee for their invaluable help in implementation.

Abstract

Secure matrix computations are one of the most useful operations for secure statistical analysis and machine learning with protecting the confidentiality of input data. Secure computations can be achieved by homomorphic encryption, supporting meaningful operations over encrypted data. Recently, several packing methods were proposed for secure matrix multiplications. In this thesis, we apply the ring-based somewhat homomorphic encryption (the BGV scheme proposed by Brakerski, Gentry, and Vaikuntanathan and the BV scheme proposed by Brakerski and Vaikuntanathan) to secure matrix multiplications. To reduce both the ciphertext size and the computation cost, we propose a new method to pack a matrix into a single ciphertext so that it also enables efficient matrix multiplication over the packed ciphertexts. Our packing method generalises Yasuda et al.'s methods (Security Comm. Networks 2015 and ACISP 2015), which are for secure inner product. In order to make our methods more efficient, we also give some modifications for secure matrix multiplication with large size and entries. 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. 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. We also give implementation details and comparison of our method over BV scheme.

As another work, we focus on the problem of performing secure predictive analysis, such as principal component analysis (PCA) and linear regression, through *exact* arithmetic over encrypted data. We improve the plaintext structure of Lu et al.'s protocols (from NDSS 2017), by switching over from *linear* array arrangement to a *two*-dimensional hypercube. This enables us to utilize the SIMD (Single Instruction Multiple Data) operations to a larger extent, which results in improving the space and time complexity

by a factor of matrix dimension. We implement both Lu et al.'s method and ours for PCA and linear regression over `HElib`. In particular, we show how to choose the optimal parameters of the BGV scheme for both methods. For example, our experiments show that our method takes 45 seconds to train a linear regression model over a dataset with 32k records and 6 numerical attributes, while Lu et al.'s method takes 206 seconds.

Table of contents

List of tables

List of figures

Chapter 1

Introduction

The development of cloud computing in recent years allows users to outsource their data in the cloud. However, security and privacy concerns for both consumers and businesses have risen at the same time. Since the publication of the RSA encryption scheme [67] in 1978, computations on encrypted data (without decryption) have attracted much attention from various communities. This encryption scheme provides only homomorphic multiplication (multiplications over encrypted data). Later, so many encryption schemes like BGN [6], Paillier [65], ElGamal encryption [29] schemes and others [22, 31, 61, 23, 63] were proposed, but all of them were either homomorphic multiplicative or homomorphic additive. However, most of the encryption schemes proposed for extending the basic model of homomorphic encryption had a lack of flexibility and supported only a few operations on encrypted data. Later, the first fully homomorphic encryption (FHE) scheme that supports arbitrary computations on encrypted data was constructed by Gentry in 2009, and this FHE scheme is also known as "Holy Grail of Cryptography". This novel encryption/decryption method not only helps to prevent malicious parties from accessing private information but also makes it possible to develop a rich collection of practical secure applications. After Gentry's breakthrough, a number of FHE schemes have been proposed and improved. Homomorphic encryption schemes have many applications, such as protocols for electronic voting schemes [3, 18, 20, 21], computational private information retrieval (PIR) schemes [53], and private matching [33]. However, currently known FHE schemes are yet impractical (e.g., it is reported in a recent work [16] 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 [65] and the BGN scheme [6] are practical, but their functionality is minimal. The Paillier scheme (resp., the BGN scheme) can support only additions (resp., additions and one-depth multiplication). 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 much 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 [62, 37, 73, 5, 75, 15]). 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.

Costache and Smart [19] compared several features of ring-based SHE schemes such as the BGV [10], FV [30], YASHE [7], and NTRU [56] schemes (see also [54] for a comparison of FV and YASHE). They also showed that the BGV scheme is more efficient for large plaintext space than other schemes. In this thesis, we choose to use the BGV scheme [10] and BV scheme proposed by Brakerski and Vaikuntanathan [12] as an alternative to the BGV scheme for secure multiplications among matrices with large entries. The security of these scheme relies on a simplified version of ring-LWE (Learning with Errors) assumption [10].

Matrix computations 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 3.1 below for an image of our goal). At present, ring-based leveled FHE schemes, such as BGV [10], FV [30], YASHE [7], and NTRU [56], are efficient and useful. BGV and FV schemes are based on the ring-LWE (learning with errors) assumption [59], and YASHE is a variant of NTRU. Costache and Smart [19] compared features of such schemes (see also [54] for a comparison), and showed that the BGV scheme is more efficient for large plaintext space than other schemes. Hence we use `HElib` [46] (Recently, `HElib` has been improved for efficiency [47]) as a software library in our implementation of the BGV scheme. For secure matrix computation, matrix-vector multiplication is proposed in `HElib` (see [45] for its manual). Recently, Lu et al. [57] slightly modified the matrix-vector multiplication for secure statistical analysis over `HElib`. As an individual work, Duong et al. [28] proposed

efficient methods for secure *single* matrix multiplication over ring-based SHE schemes. Later, Wang et al. [69] modified Duong et al.'s methods for flexible matrix computation, but their modification is much less efficient for matrices of larger size.

1.1 Related Works

Over the BV scheme, Lauter et al. [62] proposed a method to pack an integer of large size into a single ciphertext so that it enables us to compute secure sums and products over the integers efficiently. After that, Yasuda et al. [74] proposed a new packing method for secure multiple inner products, which can be applied to secure Hamming distance and pattern matching computations. While their packing method is efficient only for vectors with small size entries, they modified their method for large size entries [75]. Their modified method can be applied to secure statistical analysis such as covariance and correlation between two variables with integers of practical size. Later, Wang et al. [69] proposed a packing method for secure single matrix multiplication but their packing method needs very large parameters. For secure matrix computations, matrix-vector multiplication is proposed in `HElib` (see [45] for its manual). Recently, Lu et al. [57] slightly modified the matrix-vector multiplication for secure statistical analysis over `HElib`.

1.2 Our Contributions

In this thesis, first, we propose several packing methods for secure single matrix multiplication, using the idea of Yasuda et al.'s methods [74, 75]. More specifically, for matrices with binary entries, our method is based on [74], and for non-binary entries, it is based on [75]. Our main ingredient for packing a matrix $\mathbf{A}$ is to deploy the entries of $\mathbf{A}$ into a single polynomial by using the packing methods of [74, 75], and then encrypt the polynomial by the SHE scheme. Our matrix packing method requires only one homomorphic multiplication over the SHE scheme for secure matrix multiplication. Later we generalise our packing method from single multiplication to multiple multiplications.

For a secure multiplication between two matrices $\mathbf{A}$ and $\mathbf{B}$, a main ingredient is to pack $\mathbf{A}$ and $\mathbf{B}$ into two types of polynomial over R, and then encrypt the polynomials (method of [69] is following almost the same steps but they are packing $\mathbf{A}$ and $\mathbf{B}$ with the same type of polynomial). Homomorphic property of the ring-based scheme enables to compute all the entries of $\mathbf{A} \times \mathbf{B}$ over packed ciphertexts.

Our methods have another obstacle that it requires large t and n for matrices with large size and entries, which makes the BV scheme very slow. To address the obstacle, we give several modifications for efficiency. In particular, for large entries, we split the plaintext parameter t as $t = \prod_{i=1}^{k} t_i$ with small t_i for some k. Then, we encrypt a message modulo every t_i so that after decryption we can recover the original message from every message modulo t_i using CRT (Chinese Remainder Theorem) over the integers. Different from the double-CRT representation in [36] for SIMD (single instruction multiple data), our CRT method just splits the plaintext space R_t into small spaces R_{t_i}. However, this method requires k ciphertexts for encrypting a message. Our modifications enable us to flexibly select parameters of the BV scheme for both enough security and efficiency. For example, our the method of [28] took about 7.27 seconds for secure multiplication between two matrices with 16×16 size and 10-bit entries but our method of [60] took only about 0.50 (resp., 1.02) seconds due to our modifications for 32×32 size with 16-bit (resp., 32-bit) entries (our implementation level and security level of chosen parameters seem almost same as in [28]).

We generalize our methods of [28] 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 [28] is to pack $\mathbf{A}$ and $\mathbf{B}$ into two types of a 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 [28], 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 [28] for $\mathbf{A}_\ell$. However, 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 [37] 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 the advantages and limitations of our method.

1.3 Introduction to Public Key Cryptography

In this section, we introduce the public key encryption algorithm. For more detailed description, see [64].

Definition 1 (Public Key Encryption)**.** *A public key encryption scheme has three main algorithms: Key generation (KeyGen), Encryption (Enc) and Decryption (Dec) algorithm. First, we recall some important cryptographic notations:*

- *A message space set is denoted by M, which is the message to be handled. An element of M is called plaintext message or just a plaintext.*
- *A ciphertext space is denoted by C, an encryption of the plaintext message. A component of C is called ciphertext.*
- *A keyspace is denoted by K, which is the range of all possible values of the key.*

These algorithms have the following properties:

- $(\mathsf{pk}, \mathsf{sk}, M, C, R) \leftarrow$*KeyGen*$(1^\lambda)$*. This is a probabilistic algorithm which output a key pair public and secet key* $(\mathsf{pk}, \mathsf{sk})$ *for a given security parameter* λ *as:*

$$KeyGen = (\mathsf{pk}, \mathsf{sk}) \in K.$$

The public key is known to the public and secret key is kept private by the owner.

- $\mathsf{c} \leftarrow$*Encrypt*(m, pk, r)*. Takes a plaintext* $m \in M$ *and a public key* pk *and outputs the ciphertext* $\mathsf{c} \in C$ *denoted by:*

$$Enc(m, \mathsf{pk}) = \mathsf{c}.$$

- $m \leftarrow$*Decrypt*$(\mathsf{c}, \mathsf{sk})$*. This is a deterministic algorithm which takes the ciphertext* c *and secret key* sk *as inputs and outputs the plaintext* $m \in M$ *as:*

$$Dec(\mathsf{c}, \mathsf{sk}) = m.$$

The following probability statement give the correctness of a public key encryption scheme

$$\Pr[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{KeyGen}(1^\lambda), m \leftarrow M, r \leftarrow R, \mathsf{c} \leftarrow \mathrm{Enc}(m, \mathsf{pk}) : \mathrm{Dec}(\mathsf{c}, \mathsf{sk}) = m] = 1.$$

Chapter 2

Survey on Homomorphic Encryption Schemes

In this chapter, we introduce a basic definition of a homomorphic encryption scheme under addition and multiplication, and then we present some very well-known homomorphic encryption schemes.

Definition 2. *Let $\sigma = (P, C, K, E, D)$ be an public key encryption scheme, where P, C, K, E and D are the plaintext space, ciphertext space, keyspace, encryption algorithm and decryption algorithm respectively. Assume that the plaintexts and ciphertexts form an additive group $(P, +)$ (resp., multiplicative group $(P, \times)$) and $(C, \oplus)$ (resp., multiplicative group $(C, \otimes)$), respectively. Let $E : P \to C$ be an encryption algorithm, and $D : C \to P$ the corresponding decryption algorithm. The public key cryptosystem σ is homomorphic under addition and multiplication, if*

$$D(E(a) \oplus E(b)) = a + b$$

and

$$D(E(a) \otimes E(b)) = a \times b,$$

for all $a, b \in P$.

Basically, in this chapter, we present partial homomorphic encryption schemes (which supports either homomorphic addition or multiplication), somewhat homomorphic en-

cryption schemes (which supports limited number of homomorphic addition and multiplication), leveled homomorphic encryption schemes (supports any number of homomorphic operations but with fixed depth) and fully homomorphic encryption schemes (can operate any complex functions homomorphically).

2.1 Partial Homomorphic Encryption

In this section, we review some well known partial homomorphic encryption schemes.

2.1.1 Goldwasser-Micali Encryption Scheme

The Goldwasser-Micali (GM) encryption scheme [41] is a public key encryption algorithm constructed by Shafi Goldwasser and Silvio Micali which has homomorphic property.

KeyGen: The GM encryption scheme generates the modulus same as the RSA cryptosystem. Alice generates two distinct, random and independent large primes p and q, such that $p \equiv q \equiv 3 \mod 4$ and computes $N = pq$. She then finds some non residue a as

$$a_p^{(p-1)/2} \equiv -1 \mod p, \quad a_q^{(q-1)/2} \equiv -1 \mod q,$$

where $a_p := a \mod p$ and $a_q := a \mod q$. Set a public key $\mathsf{pk} = (a, N)$ and a secret key $\mathsf{sk} = (p, q)$.

Encryption: For a given message $m \in \mathbb{Z}$, first encode m as a string of bits $(m_1, \cdots, m_n)$. Bob chooses a random value b_i from the group units modulo N and then encrypts m_i as

$$\mathsf{c}_i \equiv b_i^2 \cdot a^{m_i} \mod N, \quad \text{when } (b_i, N) = 1.$$

Consider $\mathsf{ct} = (\mathsf{c}_1, \cdots, \mathsf{c}_n) \in (\mathbb{Z}_N)^n$ as a whole ciphertext.

Decryption: For a given ciphertext $\mathsf{ct} = (\mathsf{c}_1, \cdots, \mathsf{c}_n)$, check whether c_i is a quadratic residue (Compute $x_p := x \mod p$ and $x_q := x \mod q$ and if $x_p^{(p-1)/2} \equiv 1 \mod p$ and $x_q^{(q-1)/2} \equiv 1 \mod q$, then x is a quadratic residue modulo $N = p \cdot q$); if so, $m_i = 0$, otherwise $m_i = 1$ and then one can recover the message $m = (m_1, \cdots, m_n)$.

Homomorphic Operations: Let c_1 and c_2 are the encryption of bits m_1 and m_2, then $\mathsf{c}_1 \cdot \mathsf{c}_2 \mod N$ will be encryption of $m_0 \oplus m_1$, where $\oplus$ denotes the addition modulo N. Assume that

$$\mathsf{c}_0 \equiv b_0^2 \cdot a^{m_0} \mod N, \quad \mathsf{c}_1 \equiv b_1^2 \cdot a^{m_1} \mod N,$$

then we have

$$\mathsf{c}_0 \cdot \mathsf{c}_1 \equiv (b_0^2 \cdot a^{m_0}) \cdot (b_1^2 \cdot a^{m_1}) \mod N \equiv (b_0 b_1)^2 \cdot a^{m_0+m_1} \mod N.$$

When $m_0 + m_1$ is either 0 or 1, we have $m_0 + m_1 = m_0 \oplus m_1$. When $m_0 = m_1 = 1, m_0 + m_1 = 2$ and $\mathsf{c}_0 \cdot \mathsf{c}_1 \mod N$ is a quadratic residue and thus it is an encryption of 0. In this case, we have $m_0 \oplus m_1 = 1 \oplus 1 = 0$ as well.

Correctness: The correctness of the scheme is as follows:

- If $m_i = 0$, then $\mathsf{c}_i \equiv a_i^2 \mod N$ is a quadratic residue modulo N (and so modulo p and q also).
- If $m_i = -1$, then $\mathsf{c}_i \equiv a_i^2 \mod N$ is a quadratic non-residue modulo N (or modulo p and q).

Security: The GM encryption scheme is IND-CPA secure provided quadratic residue (QR) problem is hard (see [52] for provable security notions).

2.1.2 ElGamal Encryption Scheme

The ElGamal encryption scheme [29] is a public key encryption scheme based on the Diffie-Hellman key exchange developed by Taher Elgamal in 1985. The ElGamal encryption scheme can be defined over any cyclic group G.

KeyGen: Given a cyclic group G, of order q, with generator g. Alice chooses a random $x \in \{1, \cdots, q-1\}$ and then computes $h = g^x$. Set a public key and a secret key as $\mathsf{pk} = (G, q, g, h)$ and $\mathsf{sk} = x$, respectively.

Encryption: For a given message $m \in G$, choose a random number $r \in \{1, \cdots, q-1\}$ and compute $c_1 = g^r$ and $c_2 = ms$ with $s = h^r$. Bob sends ciphertext $\mathsf{ct} = (c_1, c_2) = (g^r, m \cdot h^r)$ to Alice.

Decryption: For a given ciphertext $\mathsf{ct} = (c_1, c_2)$, Alice computes $t = c_1^x$ and then computes $m = c_2 \cdot t^{-1}$.

Homomorphic Operations: Given two ciphertexts $\mathsf{ct} = (c_1, c_2)$ and $\mathsf{ct}' = (c_1', c_2')$, homomorphic multiplication can be computed as component-wise multiplication

$$\mathsf{ct} \otimes \mathsf{ct}' = (c_1 c_1', c_2 c_2') = (g^{r_1} g^{r_2}, (m_1 h^{r_1})(m_2 h^{r_2})) = (g^{r_1+r_2}, (m_1 m_2) h^{r_1+r_2})$$

which is the ciphertext corresponding to $m_1 m_2$.

Correctness: The correctness of the scheme can be achieved as follows:

$$c_2 \cdot t^{-1} = m \cdot h^r \cdot g^{-xr} = m \cdot g^{xr} \cdot g^{-xr} = m.$$

Security: The Elgamal encryption scheme is IND-CPA secure assuming the DDH (Decisional Diffie-Hellman) poroblem [4] is hard in G.

2.1.3 Paillier Encryption Scheme

The Paillier encryption scheme [65] is also a probabilistic public key scheme invented by Pascal Paillier in 1999. The Paillier encryption scheme contains following algorithms:

KeyGen: Generate two random large primes p and q, such that

$$\gcd(pq, (p-1)(q-1)) = 1$$

and compute

$$N = pq, \ \lambda = \mathrm{lcm}(p-1, q-1) \text{ and } \ \mu = \lambda^{-1} \mod N,$$

where lcm denotes the least common multiple. Set a public key as $\mathsf{pk} = (N, g)$ with $g = N + 1$ and a secret key as $\mathsf{sk} = (\lambda, \mu)$.

Encryption: For a given message $m \in \mathbb{Z}_N$, choose a random $r \in \mathbb{Z}_N^*$ and compute ciphertext as

$$\mathsf{ct} = (N+1)^m \cdot r^N \mod N^2.$$

Decryption: Let $\mathsf{ct} \in \mathbb{Z}_{N^2}^*$ be a ciphertext to decrypt. One can recover plaintext as

$$m = \frac{L(c^\lambda \mod N^2)}{L(g^\lambda \mod N^2)} \mod N \text{ with } L(u) = \frac{u-1}{N} \text{ for } u \in \mathbb{Z}_{N^2}.$$

Homomorphic operations: For two given ciphertexts $\mathsf{ct}_1 = g^{m_1} r_1^N$ and $\mathsf{ct}_2 = g^{m_2} r_2^N$, homomorphic addition can be computed as

$$\begin{aligned} \mathsf{ct}_1 \oplus \mathsf{ct}_2 =& (g^{m_1} \cdot r_1^N)(g^{m_2} \cdot r_2^N) \mod N^2 \\ =& g^{m_1+m_2}(r_1 r_2)^N \mod N^2 \end{aligned}$$

which is encryption of $m_1 + m_2$ and scalar multiplication can be computed as

$$\begin{aligned} E(m_1, \mathsf{pk})^{m_2} =& (g^{m_1} r_1^N)^{m_2} \mod N^2 \\ =& g^{m_1 m_2}(r_1^{m_2})^N \mod N^2 \\ =& E(m_1 m_2, \mathsf{pk}) \end{aligned}$$

which is encryption of $m_1 m_2$.

Correctness: Decryption process works because of Carmichael's theorem,

$$\begin{aligned} \lambda(N^2) =& lcm(\lambda(p^2), \lambda(q^2)) \\ =& lcm(\lambda(p^2 - p), \lambda(q^2 - q)) \\ =& pq \cdot \mathrm{lcm}(p-1, q-1) \\ =& N\lambda. \end{aligned}$$

We have $r^{N\cdot\lambda} \equiv 1 \mod N^2$. Therefore,

$$\begin{aligned} L(c^{\lambda} \mod N^2) \cdot \mu =& L((N+1)^{m\cdot\lambda} \cdot r^{N\cdot\lambda} \mod N^2) \cdot \mu \\ =& L((N+1)^{m\cdot\lambda} \mod N^2) \cdot \mu \\ =& L(1 + N \cdot m \cdot \lambda \mod N^2) \cdot \mu = m \cdot \lambda \cdot \mu \equiv m \mod N. \end{aligned}$$

Security: The Paillier encryption scheme is IND-CPA assuming the Decisional Composite Residuosity Assumption (DCRA) [65].

2.2 Somewhat Homomorphic Encryption (SHE)

In this section, we present some well-known somewhat homomorphic encryption schemes such as BGN cryptosystem and Ring-LWE based somewhat homomorphic encryption etc.

2.2.1 Boneh-Goh-Nissim Encryption Scheme

The Boneh-Goh-Nissim (BGN) encryption scheme [6] was the first to allow for arbitrary number of homomorphic additions and one multiplication by keeping the ciphertext size constant. The multiplication is possible due to pairings over elliptic curves.

KeyGen: Generate a tuple (q_1, q_2, G, G_1, e), where q_1, q_2 are two distinct large primes, G is a cyclic group of order q_1q_2 and e is pairing map $e : G \times G \to G_1$. Let $N = q_1q_2$. Choose two random generators g, u from G and set $h = u^{q_2}$. Then h is a random generator of the subgroup of G of order q_1. Set a public key as $\mathsf{pk} = (N, G, G_1, e, g, h)$ and a secret key as $\mathsf{sk} = q_1$.

Encryption: Given a message $m \in \{0, \cdots, q_2 - 1\}$, pick a random $r \in \{1, \cdots, N\}$ and compute a ciphertext

$$\mathsf{ct} = g^m h^r \in G.$$

Decryption: For a given ciphertext ct, using secret key $\mathsf{sk} = q_1$, the plaintext can be recovered as discrete logarithm of ct^{q_1} to the base g^{q_1}.

Homomorphic Operations: Let $\mathsf{ct}_1 = g^{m_1}h^{r_1} \in G$ and $\mathsf{ct}_2 = g^{m_2}h^{r_2} \in G$ be two ciphertexts corresponding to messages $m_1, m_2 \in \{0, \cdots, q_2 - 1\}$, respectively. One can compute an encryption of $m_1 + m_2 \mod N$ by computing the following product

$$\mathsf{ct}_1 \cdot \mathsf{ct}_2 \cdot h^r = (g^{m_1}h^{r_1})(g^{m_2}h^{r_2})h^r = g^{m_1+m_2}h^{r_1+r_2+r},$$

for a random $r \in \{1, \cdots, N-1\}$ which is an encryption of $m_1 + m_2 \mod N$. Moreover one can operate homomorphic multiplication using bilinear map. Let

$$g_1 = e(g, g) \text{ and } h_1 = e(g, h)$$

then g_1 and h_1 are of order N and q_1, respectively. There is some $\alpha \in \mathbb{Z}$ such that $h = g^{\alpha q_2}$. To get the encryption of $m_1 m_2 \mod N$, compute

$$\begin{aligned}
\mathsf{ct} =& e(\mathsf{ct}_1, \mathsf{ct}_2)h_1^r = e(g^{m_1}h^{r_1}, g^{m_2}h^{r_2})h_1^r \in G_1 \\
=& e(g^{m_1+\alpha q_2 r_1}, g^{m_2+\alpha q_2 r_2})h_1^r = e(g, g)^{(m_1+\alpha q_2 r_1)(m_2+\alpha q_2 r_2)}h_1^r \\
=& e(g, g)^{m_1 m_2+\alpha q_2(m_1 r_2+m_2 r_1+\alpha q_2 r_1 r_2)}h_1^r = e(g, g)^{m_1 m_2}e(g, g)^{\alpha q_2(m_1 r_2+m_2 r_1+\alpha q_2 r_1 r_2)}h_1^r \\
=& e(g, g)^{m_1 m_2}e(g, g^{\alpha q_2})^{m_1 r_2+m_2 r_1+\alpha q_2 r_1 r_2}h_1^r = e(g, g)^{m_1 m_2}e(g, h)^{m_1 r_2+m_2 r_1+\alpha q_2 r_1 r_2}h_1^r \\
=& e(g, g)^{m_1 m_2}h_1^{r+m_1 r_2+m_2 r_1+\alpha q_2 r_1 r_2} \\
=& g_1^{m_1 m_2}h_1^{r'},
\end{aligned}$$

where r' is distributed uniformly in $\mathbb{Z}_N$. Thus ct is a uniformly distributed encryption of $m_1 m_2 \mod N$, but in G_1 rather than G so that we can operate only one homomorphic multiplication.

Correctness: Correctness of the scheme can be shown as follows:

$$\log_{g^{q_1}}(\mathsf{ct}^{q_1}) = \log_{g^{q_1}}((g^m h^r)^{q_1}) = \log_{g^{q_1}}((g^{q_1})^m) = m.$$

Security: The scheme is secure under the sub-group decision problem [6] for such composite order pairing parameters.

2.2.2 Lattice Based BGN Encryption

Gentry et al. [38] proposed a public key encryption scheme, similar to the BGN cryptosystem [6] whose security is based on the LWE problem. This scheme is based on the trapdoor function proposed by Gentry, Peikert and Vaikuntanathan [39].

KeyGen:
- Fix a number q, and dimension n and m.
- Select $A \in \mathbb{Z}_q^{m \times n}$ and the trapdoor matrix $T \in \mathbb{Z}^{m \times m}$ (see [1, 2] for selecting A with T in details) such that
  - A is chosen statistically close to uniform over $\mathbb{Z}_q^{m \times n}$.
  - The rows of T from a basis of the lattice $\{x \in \mathbb{Z}^m \ : \ xA = 0 \mod q\}$.
  - T is invertible and the Euclidean norm of all rows of T is bounded by $O(n \log q)$.
- Define $\phi_\beta(q)$ to be a Gaussian error distribution over $\mathbb{Z}_q$ with Gaussian error parameter $\beta = 1/\text{poly}(n)$, as used in all LWE-based system.
- Set a public key as $\mathsf{pk} = A$ and a secret key as $\mathsf{sk} = T$.

Encryption: For a given plaintext matrix $M \in \mathbb{Z}_2^{m \times m}$, choose a uniformly random matrix $S \in \mathbb{Z}_q^{n \times m}$ and an error matrix with small entries $X \leftarrow \phi_\beta(q)^{m \times m}$ and compute the ciphertext

$$C \leftarrow AS + 2X + M \mod q.$$

Decryption: Set $E = TCT^t \mod q$ and recover M as

$$M = T^{-1}E(T^t)^{-1} \mod 2.$$

Homomorphic Operations: Homomorphic addition can be easily computed as:

$$\begin{aligned} C_1 \oplus C_2 =& (AS_1 + 2X_1 + M_1) + (AS_2 + 2X_2 + M_2) \\ =& A \cdot (S_1 + S_2) + 2 \cdot (X_1 + X_2) + (M_1 + M_2) \mod q \end{aligned}$$

and homomorphic multiplication can be computed as:

$$\begin{aligned} C_1 \otimes C_2 = C_1 \cdot C_2^t &= (AS_1 + 2X_1 + M_1) \cdot (AS_2 + 2X_2 + M_2)^t \\ &= A \cdot \underbrace{(S_1 \cdot C_2^t)}_{S} + 2 \cdot \underbrace{(X_1 \cdot (2 \cdot X_2 + M_2) + M_1 \cdot X_2^t)}_{X} \\ &+ \underbrace{M_1 \cdot M_2^t}_{M} + \underbrace{(2 \cdot X_1 + M_1) \cdot S_2^t}_{S'} \cdot A^t \mod q. \end{aligned}$$

The last equation is of the form $C = A \cdot S + 2 \cdot X + M + S' \cdot A^t$, which can be decrypted as long as the entries of $T \cdot (2 \cdot X + M) \cdot A^t$ are enough small. This scheme allows us to perform one multiplication and a limited number of additions on encrypted data.

Correctness: Recall that $TA = 0 \mod q$ and therefore $TCT^t = T(2X + M)T^t \mod q$. Moreover, all the entries of $T(2X + M)T^t$ are smaller than q (since the entries of matrices X and T are small) then we have the equality over the integers $E = (TCT^t \mod q) = T(2X + M)T^t$, and hence

$$T^{-1}E(T^t)^{-1} = T^{-1}T(2X + M)T^t(T^t)^{-1} = 2X + M \equiv M \mod 2$$

as long as all the entries of $T(2X + M)T^t$ are smaller than $q/2$.

Security: The IND-CPA security of this scheme can be reduced to the decision version of the LWE assumption [66].

2.2.3 Ring-LWE based Somewhat Homomorphic Encryption

In this section, we briefly present the construction and the homomorphic correctness of the SHE scheme proposed by Brakerski and Vaikuntanathan [12] (simply called the BV scheme). This scheme needs the following parameters:

n : an integer of 2-power defining the base ring $R = \mathbb{Z}[x]/(f(x))$ with the cyclotomic polynomial $f(x) = x^n + 1$ of degree n.

q : a prime number with $q \equiv 1 \bmod 2n$ defining the ciphertext space $R_q = R/qR = \mathbb{Z}_q[x]/(x^n+1)$. According to [11], the condition $q \equiv 1 \bmod 2n$ is not necessary for construction, but for security.

t : a positive integer with $t < q$ defining the plaintext space $R_t = R/tR = \mathbb{Z}_t[x]/(x^n+1)$.

σ : the parameter defining a discrete Gaussian error distribution $\chi = D_{\mathbb{Z}^n,\sigma}$ over $\mathbb{Z}^n$ with mean 0 and standard deviation σ.

Here we identify elements of $\mathbb{Z}^n$ as elements of R by $(a_0, \ldots, a_{n-1}) \mapsto \sum_{i=0}^{n-1} a_i x^i$.

Construction of BV Scheme

Here we present the construction of the BV scheme from [62, Section 3.2].

KeyGen: Choose $s \leftarrow \chi$, sample $p_1 \in R_q$ (uniformly) and an error $e \leftarrow \chi$. Set a public key $\mathsf{pk} = (p_0, p_1)$ with $p_0 = -(p_1 s + te)$ and a secret key $\mathsf{sk} = s$.

Encryption: For a message $m \in R_t$, sample $u, f, g \leftarrow \chi$ and compute

$$\mathrm{Enc}(m, \mathrm{pk}) = (c_0, c_1) = (p_0 u + tg + m, p_1 u + tf) \in (R_q)^2 \tag{2.1}$$

as a fresh ciphertext, where m is considered as an element of R_q since $t < q$.

Homomorphic Operations: Let $\mathrm{ct} = (c_0, \ldots, c_\xi)$ and $\mathrm{ct}' = (c'_0, \ldots, c'_\eta)$ be the two ciphertexts. The homomorphic addition "$\oplus$" is computed by component-wise addition (we pad with zeros if $\xi \neq \eta$)

$$\mathrm{ct} \oplus \mathrm{ct}' = (c_0 + c'_0, \ldots, c_{\max(\xi,\eta)} + c'_{\max(\xi,\eta)}) \in R_q^{\max(\xi,\eta)+1}.$$

The homomorphic multiplication "$*$" is defined as $\mathrm{ct} * \mathrm{ct}' = (\tilde{c}_0, \ldots, \tilde{c}_{\xi+\eta})$ with

$$\sum_{i=0}^{\xi+\eta} \tilde{c}_i z^i = \left(\sum_{i=0}^{\xi} c_i z^i\right)\left(\sum_{j=0}^{\eta} c'_j z^j\right) \in R_q[z],$$

where z is just a symbolic variable. In particular, homomorphic multiplication makes the lengths larger so that we define the decryption of any finite length of the a ciphertext.

Decryption: For a ciphertext $\mathrm{ct} = (c_0, c_1, \ldots, c_\xi)$, the decryption is computed by $\mathrm{Dec}(\mathrm{ct}, \mathrm{sk}) = [\tilde{m}]_q \bmod t$, where $\tilde{m} = \sum_{i=0}^{\xi} c_i s^i \in R$. This can be simply written as $[\langle \mathsf{ct}, \mathbf{s} \rangle]_q \bmod t$ with the secret key vector $\mathbf{s} = (1, s, s^2, \ldots)$.

Correctness: Correctness for the ciphertext $\mathsf{ct} = (c_0, c_1)$ can be followed as:

$$\begin{aligned}
\langle (c_0, c_1), (1, s) \rangle =& c_0 + c_1 s \\
=& (p_0 u + tg + m) + (p_1 u + tf)s \\
=& (-(p_1 s + te)u + tg + m) + (p_1 u + tf)s \\
=& -p_1 us - teu + tg + m + p_1 us + tfs \\
=& m + t(g + us - eu). \\
=& m \mod t.
\end{aligned}$$

Security: The security of the BV scheme relies on the following polynomial-LWE assumption, which is a simplified version of the ring-LWE assumption of Lyubashevsky, Peikert and Regev [59] (the following assumption is independent of the parameter t, see [12, Proposition 11] for details.

Definition 3 (Polynomial-LWE)**.** *Given (n, q, t, σ), the polynomial-LWE assumption is that it is infeasible to distinguish the following two distributions:*

1. *One samples (a_i, b_i) uniformly from $(R_q)^2$.*
2. *One first draws $s \leftarrow \chi = D_{\mathbb{Z}^n, \sigma}$ uniformly and then samples $(a_i, b_i) \in (R_q)^2$ by sampling $a_i \leftarrow R_q$ uniformly, $e_i \leftarrow \chi$ and setting $b_i = a_i s + e_i \in R_q$.*

Homomorphic Correctness of BV Scheme

It follows from [62, Theorem 3.3] that homomorphic operations over ciphertexts correspond to the ring structure of the plaintext space R_t. Specifically, for ciphertexts $\mathrm{ct}_1, \mathrm{ct}_2$ of plaintexts $m_1, m_2 \in R_t$, we have

$$\begin{cases} \mathrm{Dec}(\mathsf{ct}_1 \oplus \mathsf{ct}_2, \mathsf{sk}) &= m_1 + m_2, \\ \mathrm{Dec}(\mathsf{ct}_1 \otimes \mathsf{ct}_2, \mathsf{sk}) &= m_1 \times m_2, \end{cases}$$

if errors in operated ciphertexts $\mathsf{ct}_1 \oplus \mathsf{ct}_2$ and $\mathsf{ct}_1 \otimes \mathsf{ct}_2$ are small (see Lemma 4 below for details). For example, consider a fresh ciphertext ct of a message $m \in R_t$ given by Equation (2.1). If the value $m + t(g + sf - ue)$ does not wrap around mod q (all errors $e, f, g, u \leftarrow \chi$ must be sufficiently small), we have

$$[\langle \mathsf{ct}, \mathbf{s} \rangle]_q = m + t(g + sf - ue) \in R. \tag{2.2}$$

Then we can recover the correct plaintext m by decryption algorithm. The following lemma gives us the condition for the homomorphic correctness of the BV scheme. This lemma enables us to choose parameters (n, q, t, σ) for avoiding decryption failure.

Lemma 4 (Condition for successful decryption)**.** *For a given ciphertext* ct*, the decryption* $Dec(\mathsf{ct}, \mathsf{sk})$ *recovers the correct plaintext if it satisfies* $\|\langle \mathsf{ct}, \mathbf{s} \rangle\|_\infty < \frac{q}{2}$*. Here for* $a = \sum_{i=0}^{n-1} a_i x^i \in R$ *let* $\|a\|_\infty = \max |a_i|$ *denote the* ∞*-norm of its coefficient representation.*

2.2.4 YASHE Scheme

In this section, we review the YASHE (Yet Another Somewhat Homomorphic Encryption) scheme [7] which is scale-invariant so that it avoids the *modulus switching* of the BGV scheme [10] and purely based on the ring learning with errors (RLWE) assumption.

For a given security parameter λ, fix a positive integer d and modulus q that define $R = \mathbb{Z}[x]/\phi_d(x)$ as the ring of polynomials with integer coefficients modulo the d-th cyclotomic polynomial $\phi_d(x) \in \mathbb{Z}[X]$ and t with $1 < t < q$, and distributions $\chi_{\text{key}}, \chi_{\text{err}}$ on R. The message space is $R/tR = \mathbb{Z}_t[x]/\phi_d(x)$. Now, we recall two functions $P_{w,q}$ and $D_{w,q}$ from [7] which are the generalization of the PowersofTwo and the BitDecomp, respectively.

Fix a positive integer $w \geq 1$ for a radix-w system. Let $l_{w,q} = \lfloor \log_w(q) \rfloor + 2$, then a non-negative integer $w < q$ can be written as $\sum_{i=0}^{l_{w,q}-2} z_i w^i$, where z_i's are integers such that $0 \leq z_i \leq w$. Moreover, it can be uniquely written as $\sum_{i=0}^{l_{w,q}-1} z_i w^i$ for $z \in (-q/2, q/2]$ with $z_i \in (-w/2, w/2]$. Therefore, an element $x \in R$ with coefficient in $(-q/2, q/2]$ can be written as $\sum_{i=0}^{l_{w,q}-1} x_i w^i$, where $x_i \in R$ with coefficients in $(-w/2, w/2]$. With these

notations, we define:

$$D_{w,q} : R \to R^{l_{w,q}},\ x \mapsto \Big([x_0]_w, [x_1]_w, \cdots,\ [x_{l_{w,q}-1}]_w\Big) = ([x_i]_w)_{i=0}^{l_{w,q}-1} \text{ and}$$
$$P_{w,q} : R \to R^{l_{w,q}},\ x \mapsto \Big([x]_q, [xw]_q, \cdots, [xw^{l_{w,q}-1}]_q\Big) = \Big([xw^i]_q\Big)_{i=0}^{l_{w,q}-1}.$$

Key Generation: KeyGen$(d, q, t, \chi_{\text{key}}, \chi_{\text{err}}, w)$

1. Sample $f', g \leftarrow \chi_{\text{key}}$ and let $f = [tf' + 1]_q$.
2. If f is not invertible modulo q then choose a new f'.
3. Compute $f' \in R$ modulo q and set $h = [tgf^{-1}]_q$.
4. Sample $\vec{e}, \vec{s} \leftarrow \chi_{\text{err}}^{l_{w,q}}$, compute $\vec{\gamma} = [P_{w,q}(f) + \vec{e} + h \cdot \vec{s}\,]_q \in (R/qR)^{l_{w,q}}$.
5. Output$(\mathsf{pk}, \mathsf{sk}, \mathsf{evk}) = (h, f, \vec{\gamma})$,

where evk is an evaluation key which we use to refresh the ciphertext through $\mathsf{KeySwitch}$ algorithm.

Encryption: Encrypt(m, pk)

1. For a given message $m \in R/tR$, choose $[m]_t$ as its representative.
2. Sample $s, e \leftarrow \chi_{\text{err}}$.
3. Output the ciphertext $\mathsf{ct} = [\lfloor q/t \rfloor [m]_t + e + \mathsf{pk} \cdot s]_q \in R/qR$.

Decryption: Decrypt a given ciphertext ct with the secret key sk as:

$$m \leftarrow \left[\left\lfloor \frac{t}{q} \cdot [\mathsf{sk} \cdot \mathsf{ct}]_q \right\rceil\right]_t \in R/tR.$$

Key Switching: $\mathsf{keyswitch}(\tilde{\mathsf{ct}}_{\text{Mult}}, \text{evk})$

1. Output the ciphertext $[\langle D_{w,q}(\tilde{\mathsf{ct}}_{\text{Mult}}), \text{evk}\rangle]_q$.

The key switching algorithm transforms an intermediate encryption into a ciphertext that can be decrypted with f itself. The evaluation key is $\text{evk} = [P_{w,q}(f) + \vec{e} + h \cdot \vec{s}]_q$, where $\vec{e}, \vec{s} \leftarrow \chi_{\text{err}}^{l_{w,q}}$ are vectors of polynomials sampled from the error distribution χ_{err}.

This key is a vector of quasi-encryption of the secret key f under its corresponding public key. It is required for the homomorphic multiplication and is therefore made public. This means, we need to make a circular security assumption [13], namely that the scheme is still secure even given that evk is publicly known.

Homomorphic Addition For the two given ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2 \in R/qR$ encrypting the messages m_1, m_2, respectively, homomorphic addition can be performed as:

$$\mathsf{ct}_{\text{add}} = [\mathsf{ct}_1 + \mathsf{ct}_2]_q.$$

Homomorphic Multiplication For the two given ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2 \in R/qR$ encrypting the messages m_1, m_2, respectively, homomorphic multiplication can be performed as:

$$\mathsf{ct}_{\text{Mult}} = \mathsf{KeySwitch}(\tilde{\mathsf{ct}}_{\text{Mult}}, \text{evk}), \text{ where } \tilde{\mathsf{ct}}_{\text{Mult}} = \left[\left\lfloor \frac{t}{q} \cdot \mathsf{ct}_1 \cdot \mathsf{ct}_2 \right\rceil\right]_t.$$

2.2.5 Gentry's SHE scheme

Gentry proposed the first GGH-type encryption scheme in his PhD thesis [35], where GGH scheme was presented originally by Goldreich et al. [40]. In this scheme, Ideal I is considered as a $\mathbb{Z}$ module which has a basis matrix B_I, where all elements in the ideal are given by vectors of the form $B_I \cdot \mathbf{x}$ with $\mathbf{x} \in \mathbb{Z}^n$. The fundamental parallelpiped associated to B_I is the domain, consisting of the vectors $B_I \cdot \mathbf{x}$ where $\mathbf{x} \in (-1/2, 1/2]^n$. Reduction of a vector $\mathbf{x} \in \mathbb{Z}^n$ modulo the basis is defined to be the element in the fundamental parallelpiped which is equivalent to the input vector $\mathbf{x}$ under translations in the lattice and can be computed as

$$\mathbf{x} \mod I = \mathbf{x} - B_I \cdot \lfloor B_I^{-1} \cdot \mathbf{x} \rceil,$$

where $\lfloor \cdot \rceil$ rounds each coefficient vector to the nearest integer. This reduction depends on the specific basis being considered so it is always better to write $\mathbf{x} \mod B_I$ and we

denote it by $\Lambda(B_I)$. Gentry started his breakthrough work from somewhat homomorphic encryption which consists of the following algorithms:

KeyGen:

- Choose a polynomial $F(x)$ defining a number field K.
- Pick a basis B_I of an ideal I of "small" norm.
- Generate two basis B_J^{sk} and B_J^{pk} of an ideal of "large" norm.
- Choose a small integer μ.
- Public key pk consists of $(F, B_I, B_J^{\mathsf{pk}})$ and the secret key sk only includes B_J^{sk}.
- $M := \Lambda(B_I), C := \Lambda(B_J^{\mathsf{pk}}), R := [-\mu, \cdots, \mu]^n$,

where $\Lambda(B_I)$ and $\Lambda(B_J^{\mathsf{pk}})$ stand for the lattice generated by the bases B_I and B_J^{pk} respectively. Basically, B_J^{sk} will be a "nice" basis, i.e., one which consists of vectors which are nearly orthogonal, whereas B_J^{pk} will be a "horrible" basis, e.g. the HNF of the basis matrix B_j^{sk}.

Encryption:

- $i \leftarrow B_I \cdot r$.
- $\mathsf{ct} \leftarrow (m + i) \mod B_J^{\mathsf{pk}}$.

Decryption:

- $t \leftarrow \mathsf{ct} \mod B_J^{\mathsf{sk}}$.
- $m \leftarrow t \mod B_I$.

If the randomness r is small enough, and the basis B_J^{sk} is "nice" enough the decryption procedure will be correct.

Homomorphic Operations: The homomorphic property follows from the basic properties of an ideal I in a ring R, namely

- If $i_1, i_2 \in I$ then $i_1 + i_2 \in I$.
- If $i \in I$ and $r \in R$ then $r \cdot i \in I$.

Since a ciphertext is of the form $m+i$ without reduction modulo B_J^{pk}. We have that each ciphertext is of the form $m+i+j$ where $j \in J$ is chosen to reduce $m+i$ into the domain $\Lambda(B_J^{\mathsf{pk}})$. From these facts we deduce the homomorphic addition

$$\begin{aligned}\mathsf{ct}_1 \oplus \mathsf{ct}_2 :=& \mathsf{ct}_1 + \mathsf{ct}_2 \mod B_J^{\mathsf{pk}} \\ =&(m_1+i_1+j_1)+(m_2+i_2+j_2)+j_3 \\ =&(m_1+m_2)+(i_1+i_2)+(j_1+j_2+j_3) \\ =&(m_1+m_2)+i+j\end{aligned}$$

for some $i \in I$ and $j, j_3 \in J$. Homomorphic multiplication can be performed as

$$\begin{aligned}\mathsf{ct}_1 \otimes \mathsf{ct}_2 :=& \mathsf{ct}_1 \cdot \mathsf{ct}_2 \mod B_J^{\mathsf{pk}} \\ =&(m_1+i_1+j_1)\cdot(m_2+i_2+j_2)+j_3 \\ =&(m_1\cdot m_2)+(i_1m_2+i_2m_1+i_1i_2)+(j_1\cdot(m_2+i_2) \\ &+j_2\cdot(m_1+i_1)+j_1\cdot j_2+j_3) \\ =&(m_1+m_2)+i+j.\end{aligned}$$

But now the randomness of vectors i, j increases much more than addition. Operations make the resulting ciphertexts more "dirty" and once it becomes too dirty it can not be decrypted. Thus the scheme is homomorphic over the ring R, but is bounded in nature.

Security: The security of this scheme is based on Ideal Coset Problem introduced in [35].

2.2.6 NTRU-based Homomorphic Encryption

Hoffstein, Pipher and Silvermanin proposed the NTRU scheme [49] which is the first public key cryptosystem which is not based on factorization or discrete logarithm problems. NTRU is a lattice version of RSA and ECC and based on the shortest vector problem in a lattice. It became interesting after Gentry's bootstrapping operation and relinearisation technique of Brakerski and Vaikuntanathan. We take cyclotomic ring $R = \mathbb{Z}[x]/(x^n-1)$ for NTRU cryptosystem, R_p and R_q for plaintext and ciphertext space, respectively.

KeyGen Choose small elements f' and g in R and set $f = p \cdot f' + 1 \mod q$ (if f is not invertible then start from the beginning). Set a secret key sk as $f \in R$ and a public key pk as $h = pgf^{-1} \in R_q$.

Encryption For a given message $m \in R_p$, one can encrypt as

$$\mathsf{ct} = \mathrm{Enc}(m) = m + pe + hs \mod q,$$

where s and e is chosen small elements in R.

Decryption The decryption algorithm with secret key f works as follows:

$$m = \lfloor (f \cdot \mathsf{ct} \mod q) \rceil \mod p.$$

Homomorphic Property Note that

$$f \cdot \mathsf{ct} \mod q = f \cdot m + pfe + psg.$$

Suppose there will no wrap around and f, g, s and e are considered to be the small then we have

$$f \cdot \mathsf{ct} \mod q = f \cdot m + p \cdot \mathrm{small}.$$

Now it is evident that one can easily add two ciphertexts homomorphically and can be decrypted with secret key f. If we multiply two ciphertexts then the result ciphertext can decrypted with secret key f^2.

Security The security of this scheme rely on the hardness of Ring-LWE problem [59].

2.2.7 Homomorphic Encryption over Integers

Van Dijk et al [26] proposed an SHE scheme, one year after Gentry's original scheme which can achieve fully homomorphic encryption using Gentry's bootstrapping method. This construction needs mainly four parameters with the security parameter λ:

$\gamma \leftarrow O(\lambda^5)$: bit-length of the integers in public key.

$\eta \leftarrow O(\lambda^2)$: bit-length of the secret key.

$\rho \leftarrow 2 \cdot \lambda$: bit-length of the noise.

$\tau \leftarrow \gamma + \lambda$: number of integers in the public key.

KeyGen The secret key is an odd $\eta-$ bit integer:

$$p \leftarrow (2\mathbb{Z} + 1) \cap [2^{\eta-1}, 2^{\eta}).$$

For the public key, sample $x_i \leftarrow D_{\gamma,\rho}(p)$ for $i = 0, \ldots, \tau$. Relevel so that x_0 is the largest. Restart unless x_0 is odd and $\gamma_p(x_0)$ is even and public key pk as $\langle x_0, \ldots, x_\tau \rangle$.

Encryption Choose a random subset $S \subset \{1, 2, \ldots, \tau\}$ and a random integer in $(-2^{\rho}, 2^{\rho})$. Encrypt a message m as

$$\mathsf{ct} \leftarrow m + 2 \cdot t + 2 \sum_{i \in S} x_i \mod x_0.$$

Decryption Decrypt a ciphertext ct as:

$$m \leftarrow (\mathsf{ct} \mod p) \mod 2.$$

Homomorphic Property The homomorphic addition can be performed as follows:

$$\mathsf{ct}_1 \oplus \mathsf{ct}_2 = (m_1 + 2r_1 + pq_1 + m_2 + 2r_2 + pq_2) = (m_1 + m_2) + 2(r_1 + r_2) + (q_1 + q_2)q$$

and can decrypted if the noise $|m_1 + 2r_1 + m_2 + 2r_2| < p/2$ and the homomorphic multiplication can be performed as follows:

$$\mathsf{ct}_1 \otimes \mathsf{ct}_2 = (m_1 + 2r_1 + pq_1)(m_2 + 2r_2 + pq_2) = m_1 m_2 + 2(m_1 r_2 + m_2 r_1 + 2r_1 r_2) + kp$$

and can be decrypted successfully if the noise $|m_1 m_2 + 2(m_1 r_2 + m_2 r_1 + 2r_1 r_2)| < p/2$.

Security The security of the scheme depends on the hardness of the Approximate-Greatest Common Divisor (AGCD) problem [34].

2.2.8 Homomorphic Encryption over the Torus

Chillotti et al [16] proposed a fully homomorphic encryption scheme over the torus in Asiacrypt 2016. Denote $\mathbb{T}$ the real torus $\mathbb{R}/\mathbb{Z}$, the set of real numbers modulo 1 and $\mathbb{T}_n[x] = \mathbb{R}_n[x]/\mathbb{Z}_n[x] = \mathbb{R}[x] \mod x^n + 1 \mod 1$. Note that $\mathbb{T}_n[x]$ is a $\mathbb{Z}_n[x]$-module, not a ring so it does not have internal multiplication, but it has an external product with coefficients in $\mathbb{Z}_n[x]$. We also denote $\mathbb{B}_n[x]$ the subset of $\mathbb{Z}_n[x]$ with binary coefficients. Here, the message and ciphertexts are represented over the torus modulo 1 ($\mathbb{T} = \mathbb{R}/\mathbb{Z}$). Basically, TFHE can represent three plaintext spaces using TLWE, TRLWE, and TRGSW but here we briefly review the TFHE with TRLWE encryption scheme.

Parameters: A security parameter λ, and a minimal noise α to define a minimal key size n.

KeyGen A uniform random binary key $s \in \mathbb{B}_n[x]$ which defines the secret phase function $\phi_s : \mathbb{T}_n[x]^2 \to \mathbb{T}_n[x]$ such that $(a, b) \longmapsto (b - sa)$.

Encrypt (μ, s, α) Choose a uniformly random element $a \in \mathbb{T}_n[x]$, and a small Gaussian error $e \leftarrow D_{\mathbb{T}_n[x],\alpha}$ and return a ciphertext $\mathsf{ct} = (a, s \cdot a + \mu + e)$.

DecryptApprox(c, s) Return $\phi_s(\mathsf{ct}) = (s \cdot a + \mu + e - s \cdot a) = (\mu + e)$, which is close to the actual message.

DecryptRound(ct, s, M) Round $\phi_s(\mathsf{ct})$, which is nearest point in M.

Linear Combination return $\sum e_i \cdot \mathsf{ct}_i$, where $e_i \in \mathbb{Z}_n[x]$.

External Product Given a TRGSW ciphertext A of $M \in \mathbb{Z}_n[x]$ and a TRLWE ciphertext b of $\mu \in \mathbb{T}_n[x]$, compute $A \odot_\alpha b$ at precision $\alpha > 0$, which encrypts $M \cdot \mu \in \mathbb{T}_n[x], \odot_\alpha : \text{TRGSW} \times \text{TRLWE} \to \text{TRLWE}$ (see [17] for details).

2.3 Fully (Unbounded) Homomorphic Encryption

If a bounded (SHE) scheme can evaluate its decryption circuit, plus a little more, then one can bootstrap the homomorphic operations to construct a fully (unbounded) homomorphic encryption scheme. Gentry constructed the first fully homomorphic encryption (FHE)

scheme that supports arbitrary computations on encrypted data in 2009, and this FHE scheme is also known as "Holy Grail of Cryptography". This novel encryption/decryption method not only helps to prevent malicious parties from accessing private information but also makes it possible to develop a rich collection of practical and secure applications. After Gentry's breakthrough, a number of FHE schemes have been proposed and improved. However, currently known FHE schemes are yet impractical (e.g., it is reported in a recent work [16] that it takes 52 milliseconds for *single gate* bootstrapping to refresh the error in a ciphertext for unlimited operations).

2.3.1 Gentry's Scheme

From Gentry's SHE scheme [35] we see that Gentry's scheme use an ideal I of a ring R and homomorphic operation for $\mathsf{ct}_1 = m_1 + r_1 I$ and $\mathsf{ct}_2 = m_2 + r_2 I$ can be seen that

$$\mathsf{ct}_1 \oplus \mathsf{ct}_2 = (m_1 + m_2) + (r_1 + r_2)I.$$
$$\mathsf{ct}_1 \otimes \mathsf{ct}_2 = (m_1 + r_1 I)(m_2 + r_2 I) = (m_1 m_2) + (m_1 r_2 + m_2 r_1 + r_1 r_2 I)I.$$

Notice that the noise $r_1 r_2 I$ after the multiplication is dominating the noise $(r_1 + r_2)I$ after the addition more precisely, homomorphic addition doubles the noise while multiplication squares the noise and after a number of operations, the ciphertext will be very noisy and will be no longer decryptable. To resolve this issue, Gentry came up with a solution in 2009 by evaluating the decryption circuit homomorphically with ciphertext as input and output a refreshed ciphertext encrypting the same message with small noise. This procedure is known as *bootstrapping* and to keep the bootstrapping process efficient, the decryption circuit should be as simple as possible. Moreover, Gentry proposed another important process named as *squashing* to reduce the complexity of the decryption circuit. However, if SHE scheme is bootstrappable (bootstrappable means that the SHE scheme can evaluate its own decryption function) then only it can be converted into a fully homomorphic encryption. Depth of the circuit is related to the maximum number of operations. If an SHE scheme is not bootstrappable already, it can be made by "squashing" the decryption circuit.

Squashing

Gentry's bootstrapping technique is allowed only for the decryption algorithm with small depth. Gentry provide a squashing technique which used some tweeks to reduce the complexity of decryption algorithm and works as follows:

First choose a set of vectors, whose sum equals to the multiplicative inverse of $(B_J^{SK})^{-1}$. If the ciphertext is multiplied by the elements of this set, the polynomial of degree of the circuit is reduced to the level that the scheme can handle.

Bootstrapping

Bootstrapping is also known as "recryption" which is to get a "fresh" ciphertext from the noisy ciphertext encrypting the same message. If the ciphertext is not bootstrappable then we make this ciphertext bootstrappable using squashing technique. The bootstrapping procedure of Gentry's scheme works as follows:

Assume that two different public and secret key pairs $(\mathsf{pk}_1, \mathsf{sk}_1)$ and $(\mathsf{pk}_2, \mathsf{sk}_2)$ are generated. The secret keys are kept secret by the user while the public keys along with encryption of the secret key $\mathrm{Enc}_{\mathsf{pk}_1}(\mathsf{sk}_1)$ are shared with server which already has $\mathsf{c}_1 = \mathrm{Enc}_{\mathsf{pk}_1}(m)$. Since the SHE scheme is already squashed, the noisy ciphertext is decrypted homomorphically using $\mathrm{Enc}_{\mathsf{pk}_1}(\mathsf{sk}_1)$ to remove the noise and the result is encrypted using a different public key pk_2, i.e.,

$$\mathsf{c}_2 = \mathrm{Enc}_{\mathsf{pk}_2}(\mathrm{Dec}_{\mathsf{sk}_1}(\mathsf{c}_1)) = \mathrm{Enc}_{\mathsf{pk}_2}(m)$$

which has a smaller noise. The ciphertext c_2 can be decrypted using sk_2 which is kept secret by the user, i.e.,

$$\mathrm{Dec}_{\mathsf{sk}_2}(\mathrm{Enc}_{\mathsf{pk}_2}(m)) = m.$$

The security of Gentry's scheme is based on two assumed hardness problems: certain worst-case problems over ideal lattices and (average case) sparse subset-sum problem.

2.3.2 Brakerski and Vaikuntanathan's Scheme

Brakerski and Vaikuntanathan in [12] proposed a public key homomorphic encryption scheme based on LWE assumption in 2011. In spite of going into details of the scheme, we directly write the decryption algorithm (see Subsection 2.2.3 for the details of the encryption scheme) which is as follows:

$$\begin{aligned} b - \langle a, s\rangle =& (v^T r + m) - (a^T s) \\ =& (v^T r + m) - (v^T r - 2e^T r) \\ =& 2e^T r + m = m \mod 2. \end{aligned}$$

Relinearization:

Given a ciphertext $(\boldsymbol{a}, b)$, $\boldsymbol{a} = (a[1], \cdots, a[n])$, consider the linear evaluation function $f_{\boldsymbol{a},b} : \mathbb{Z}_q^n \to \mathbb{Z}_q$ defined by $f(\boldsymbol{a}, b) = b - \langle \boldsymbol{a}, \boldsymbol{x}\rangle = b - \sum_{i=1}^n \boldsymbol{a}[i]\boldsymbol{x}[i] \mod q$ with $\boldsymbol{x} = (x[1], \cdots, x[n])$. One can recover m by substituting $x = s \mod 2$ in the function $f_{\boldsymbol{a},b}(\mathbf{x})$. Note that $f_{\boldsymbol{a},b}$ is homomorphic in addition since

$$\begin{aligned} f_{(\boldsymbol{a},b)}(\boldsymbol{x}) + f_{(\boldsymbol{a}',b')}(\boldsymbol{x}) =& (b - \sum_{i=1}^n \boldsymbol{a}[i]\boldsymbol{x}[i]) + (b' - \sum_{i=1}^n \boldsymbol{a}'[i]\boldsymbol{x}[i]) \\ =& (b + b') - \sum_{i=1}^n (\boldsymbol{a}[i] + \boldsymbol{a}'[i]) \cdot \boldsymbol{x}[i] \\ =& f_{(\boldsymbol{a}+\boldsymbol{a}', b+b')}(\boldsymbol{x}). \end{aligned}$$

But, multiplication

$$\begin{aligned} f_{(\boldsymbol{a},b)}(\boldsymbol{x}) \cdot f_{(\boldsymbol{a}',b')}(\boldsymbol{x}) =& (b - \sum_{i=1}^n \boldsymbol{a}[i]\boldsymbol{x}[i]) \cdot (b' - \sum_{i=1}^n \boldsymbol{a}'[i]\boldsymbol{x}[i]) \\ =& h_0 + \sum_{i=1}^n h_i \cdot \boldsymbol{x}[i] + \sum_{1 \le i \le j \le n} h_{i,j} \cdot \boldsymbol{x}[i]\boldsymbol{x}[j], \end{aligned}$$

where $h_0 = bb', h_i = -(b\boldsymbol{a}')[i] + b'\boldsymbol{a}[i])$, and $h_{i,j} = \boldsymbol{a}[i]\boldsymbol{a}'[j] + \boldsymbol{a}[j]\boldsymbol{a}'[i]$ seems problematic. The authors overcome with a method known as *relinearisation* by changing the secret

key $\boldsymbol{s} = (\boldsymbol{s}[1], \cdots, \boldsymbol{s}[n])$ into a secret key

$$t = (\boldsymbol{s}[1], \cdots, \boldsymbol{s}[n], \boldsymbol{s}[1]\boldsymbol{s}[1], \boldsymbol{s}[1]\boldsymbol{s}[2], \cdots, \boldsymbol{s}[n]\boldsymbol{s}[n]).$$

Then $h_{\boldsymbol{a},b}(\boldsymbol{x}) = f_{(\boldsymbol{a},b)}(\boldsymbol{x}) \cdot f_{(\boldsymbol{a}',b')}(\boldsymbol{x})$ becomes linear in t and $h_{\boldsymbol{a},b}(\boldsymbol{t}) \mod 2 = m \cdot m'$.

Modulus Switching:

To achieve the fully homomorphic encryption scheme, the challenging task is to reduce the noise. The authors in [12] proposed a method called *modulus switching* to resolve this problem. Modulus switching technique changes the ciphertext $c \in \mathbb{Z}_q^n$ to a ciphertext $c' \in \mathbb{Z}_p^n$ encrypting the same message with $p \leq q$.

2.3.3 BGV Cryptosystem

In this work, we use the Ring-LWE variant of the BGV scheme [37], which is defined over the polynomial ring $\mathbf{A} = \mathbb{Z}[X]/\Phi_m(X)$, where $\Phi_m(X)$ is the m-th cyclotomic polynomial.

Plaintext Space

The plaintext space is defined by the ring $\mathbf{A}_t = \mathbf{A}/t\mathbf{A}$, where t is a prime. Under modulo t, the polynomial $\Phi_m(X)$ factors into ℓ irreducible polynomials, each of degree $s = \phi(m)/\ell$ such that $\Phi_m(X) = F_1(X) \cdot F_2(X) \cdots F_\ell(X) \pmod t$. Each factor $F_i(X)$ corresponds to a plaintext slot (each slot is isomorphic to the finite field $\mathbb{F}_{t^s}$) and the following isomorphism holds:

$$\mathbf{A}_t \simeq \mathbb{Z}_t[X]/F_1(X) \times \cdots \times \mathbb{Z}_t[X]/F_\ell(X) \simeq \mathbb{F}_{t^s} \times \cdots \times \mathbb{F}_{t^s}.$$

Therefore, a polynomial $a(X) \in \mathbf{A}_t$ can be represented as the vector $(a \bmod F_i)_{i=1}^{\ell}$. `HElib` provides high level interfaces that allow conversion between a vector of plaintext values $(a^{(i)})_{i=1}^{\ell} \in (\mathbb{F}_{t^s})^\ell$ and a polynomial $a(X) \in \mathbf{A}_t$ (native plaintext space of the BGV scheme) through encoding and decoding routines. Hence, given two polynomial encodings

$a(X) = \mathsf{Encode}\Big((a_i)_{i=1}^{\ell}\Big)$ and $b(X) = \mathsf{Encode}\Big((b_i)_{i=1}^{\ell}\Big)$, we have:

$$\begin{cases} \mathsf{Decode}(a + b \bmod (t, \Phi_m)) = (a_i + b_i \bmod (t, F_i))_{i=1}^{\ell} \\ \mathsf{Decode}(a \cdot b \bmod (t, \Phi_m)) = (a_i \cdot b_i \bmod (t, F_i))_{i=1}^{\ell} \end{cases}.$$

Each slot in the vector representation corresponds to a unique conjugacy class of $\mathbb{Z}_m^*/\langle t \rangle$. An isomorphism exists between the polynomial ring and the vector of plaintext slots, and the plaintext slots are isomorphic to one another. This imparts automorphic mappings of the form $\kappa : a(X) \to a(X^k)$, where $k \in \mathbb{Z}_m^*/\langle t \rangle$, that allow data movement among the slots.

The structure of $\mathbb{Z}_m^*/\langle t \rangle$ can be represented by a set of generators $\{f_1, \ldots, f_n\}$, where the order of f_i in $\mathbb{Z}_m^*/\langle t, f_1, \ldots, f_{i-1} \rangle$ is m_i. Each slot has a unique representative in the slot-index representative set $\mathrm{T} = \left\{ \prod_{i=1}^{n} f_i^{e_i}, 0 \leq e_i \leq m_i - 1 \right\}$ which can be indexed by the vector of exponents $(e_1, \ldots, e_n)$. Therefore, the plaintext slots can be mapped to an n-dimensional *hypercube*, whose i-th dimension is of size m_i. The i-th dimension is labelled as a *good* dimension if $m_i = \mathsf{ord}_{\mathbb{Z}_m^*}(f_i)$, otherwise it is labelled as a *bad* dimension. *Good* dimensions lead to more efficient data movement (see [44, Section 4] for more details).

Ciphertext Space

The ciphertext space is defined by vectors over the ring $\mathbf{A}_q = \mathbf{A}/q\mathbf{A}$, where q is an odd modulus that changes over homomorphic evaluation. The scheme with level parameter L is parametrized by a chain of moduli

$$q_0 < q_1 < \cdots < q_{L-1} \tag{2.3}$$

and freshly encrypted ciphertexts are defined over $\mathbf{A}_{q_{L-1}}$. Ciphertexts defined over $\mathbf{A}_{q_l}$ are called level-l ciphertexts.

The modulus q_l (also called level-l modulus) is defined as the product of $l+1$ small primes p_i of same size chosen such that $m \equiv 1 \bmod p_i$ (`HElib` provides an additional half-prime optimization. See [37, Section 3] for details). This is done so that for all i, $\Phi_m(X)$ factors linearly under modulo p_i.

For efficient arithmetic over ciphertexts, a polynomial $a(X) \in \mathbf{A}_{q_l}$ (in coefficient representation) is represented as a $(l+1) \times \phi(m)$ matrix $\mathsf{DoubleCRT}^l(a)$ (in evaluation representation), whose (i,j)-th entry is the evaluation of $a(X)$ at j-th root of $\Phi_m(X)$ modulo p_i. Addition and multiplication in $\mathbf{A}_{q_l}$ is done entry-wise modulo the appropriate primes p_i. For ease of representation, in the rest of description we ignore the double CRT representation of polynomials lying in ciphertext space and describe the scheme as if we were operating on polynomials directly.

Key Generation

Given a parameter w, a random low norm polynomial $\mathsf{sk} \in \mathbf{A}_{q_{L-1}}$ having coefficients in $\{-1, 0, 1\}$ is chosen such that its Hamming weight is exactly w. Secret key is set as $\mathsf{sk} = (1, -\mathsf{sk})$. To generate a public key, a uniformly random polynomial $a \in \mathbf{A}_{q_{L-1}}$ is chosen and a low-norm error polynomial $e \in \mathbf{A}_{q_{L-1}}$ is sampled from χ. The public key is set as $\mathsf{pk} = (a, b)$, where $b = [a \cdot \mathsf{sk} + t \cdot e]_{q_{L-1}}$.

In addition to this, the public key also has key switching matrices of the form $W[\mathsf{sk}' \to \mathsf{sk}]$ that transform a ciphertext decryptable by sk' into a ciphertext decryptable by sk. Specifically, we have $W[\mathsf{sk}^2 \to \mathsf{sk}]$ and $W[\mathsf{sk}(X^k) \to \mathsf{sk}]$, where $k \in \mathbb{Z}_m^*/\langle t \rangle$, which are used in multiplication and data movement respectively to get "canonical" ciphertext of the form $\mathbf{c} = (c_0, c_1) \in (\mathbf{A}_{q_l})^2$, that are decryptable by a secret key of the form $\mathsf{sk} = (1, -\mathsf{sk})$.

Encryption

To encrypt a polynomial $m \in \mathbf{A}_t$, a random low norm polynomial $r \in \mathbf{A}_{q_{L-1}}$ having coefficients in $\{-1, 0, 1\}$ is chosen and two low-norm error polynomials $e_0, e_1 \in \mathbf{A}_{q_{L-1}}$ are sampled from χ. The ciphertext is computed as:

$$\mathbf{c} = (c_0, c_1) = \mathsf{Enc}(m, \mathsf{pk}) = (b \cdot r + t \cdot e_0 + m, a \cdot r + t \cdot e_1) \in (\mathbf{A}_{q_{L-1}})^2.$$

Decryption

To decrypt a level-l ciphertext, we first compute the noise polynomial $m' = [\langle \mathbf{c}, \mathsf{sk} \rangle]_{q_l} = [c_0 - c_1 \cdot \mathsf{sk}]_{q_l}$. Then, the message $m \in \mathbf{A}_t$ is recovered by computing $m' \bmod t = \mathsf{Dec}(\mathbf{c}, \mathsf{sk})$.

For the decryption procedure to work, the norm of noise polynomial m' should be sufficiently smaller than q_l.

Homomorphic Operations and Noise Control

The noise associated with a ciphertext should be considerably small compared to the ciphertext modulus to successfully recover the message. But, homomorphically operating on ciphertexts leads to an increase in the noise term. The freshly encrypted ciphertext is valid w.r.t. the largest modulus q_{L-1}. When the noise term grows too much, we modulus-switch to a ciphertext that is valid w.r.t. smaller moduli to decrease the noise magnitude. As we operate on a ciphertext, we need to switch to smaller moduli until the ciphertext is defined over $\mathbf{A}_{q_0}$. Beyond this point, we can not modulus-switch any further, and if the noise grows too much now, the ciphertext will be rendered useless.

In `HElib`, every l-th level ciphertext is represented as the tuple $\mathbf{c} = ((c_0, c_1), l, \nu)$, where ν is an estimate of noise magnitude of the ciphertext. This estimate helps the library in automatically switching to lower levels when needed (for details on noise estimate, see [44, Section 3.1.4]). The homomorphic operations are defined as follows:

- *Addition*: Before *adding* ciphertexts $\mathbf{c} = ((c_0, c_1), l, \nu)$, $\mathbf{c}' = ((c'_0, c'_1), l', \nu')$ encrypting messages $m, m' \in \mathbf{A}_t$ respectively, we bring them to the same level l'' (if $l \neq l'$) by reducing the larger one modulo the smaller of the two moduli if the noise doesn't overflow, otherwise we modulus-switch the larger one to the smaller level. Then, we add the ciphertexts to get:

$$\mathbf{c}_{\mathsf{add}} = (([c_0 + c'_0]_{q_{l''}}, [c_1 + c'_1]_{q_{l''}}), l'', \nu + \nu').$$

- *Multiplication*: Given two ciphertexts $\mathbf{c} = ((c_0, c_1), l, \nu)$, $\mathbf{c}' = ((c'_0, c'_1), l', \nu')$ encrypting $m, m' \in \mathbf{A}_t$, we first perform modulus switching on them to bring their noise magnitude below a preset constant (refer [37, Appendix C.2] for details). Then, we reduce the larger one modulo the smaller of the two moduli to bring them to the same level l''. Having brought the ciphertexts to the same level, we perform their tensor product to get:

$$\mathbf{c}'_{\mathsf{mult}} = ((c_0 c'_0, c_0 c'_1 + c'_0 c_1, c_1 c'_1), l'', \nu\nu').$$

$\mathbf{c}'_{\mathsf{mult}}$ is a ciphertext decryptable by $\mathsf{sk}' = (1, -\mathsf{sk}, \mathsf{sk}^2)$. We perform key-switching on $\mathbf{c}'_{\mathsf{mult}}$ using $W[\mathsf{sk}^2 \rightarrow \mathsf{sk}]$ to get a canonical ciphertext $\mathbf{c}_{\mathsf{mult}} = ((c''_0, c''_1), l'', \nu'')$ with noise magnitude ν''. We can also multiply the ciphertext with a scalar. So, given a scalar $\alpha \in \mathbf{A}_t$ and a ciphertext $\mathbf{c} = ((c_0, c_1), l, \nu)$ encrypting $m \in \mathbf{A}_t$, we can multiply them to get the ciphertext $\mathbf{c}_{\mathsf{scalar-mult}} = ((\alpha \cdot c_0, \alpha \cdot c_1), l, \nu')$ encrypting $\alpha \cdot m \in \mathbf{A}_t$. The noise estimate ν' is computed as $\nu' = \nu \cdot \nu_\alpha$, where ν_α is the maximum norm possible for the scalar α.

- *Automorphism*: As mentioned in Section 2.3.3, data movement is made possible by automorphic mappings of the form $\kappa : a(X) \rightarrow a(X^k)$, where $k \in \mathbb{Z}^*_m/\langle t \rangle$. Given a ciphertext $\mathbf{c} = ((c_0, c_1), l, \nu)$ encrypting $m \in \mathbf{A}_t$, by applying automorphism (note that the noise does not change), we get:

$$\mathbf{c}'_{\mathsf{auto}} = ((\kappa(c_0), 0, \kappa(c_1)), l, \nu).$$

$\mathbf{c}'_{\mathsf{auto}}$ is a ciphertext decryptable by $\mathsf{sk}' = (1, -\mathsf{sk}, -\kappa(\mathsf{sk}))$. We perform key-switching on $\mathbf{c}'_{\mathsf{auto}}$ using $W[\mathsf{sk}(X^k) \rightarrow \mathsf{sk}]$ to get a canonical ciphertext $\mathbf{c}_{\mathsf{auto}} = ((c'_0, c'_1), l, \nu')$ with noise magnitude ν'.

The BGV scheme has a ring homomorphism between the plaintext and ciphertext space. Therefore, we have:

$$\left.\begin{aligned} \mathsf{Dec}(\mathbf{c}_{\mathsf{add}}, \mathsf{sk}) &= m + m' \\ \mathsf{Dec}(\mathbf{c}_{\mathsf{mult}}, \mathsf{sk}) &= m \cdot m' \\ \mathsf{Dec}(\mathbf{c}_{\mathsf{auto}}, \mathsf{sk}) &= \kappa(m) \end{aligned}\right\} \in \mathbf{A}_t.$$

In this scheme, authors introduced the technique *key switching/ modulus* reduction (which is the generalization of relinearization introduced in [12]) and improved modulus switching of [12]. These techniques refined the ciphertext much better so that a fully homomorphic encryption scheme can be achieved without bootstrapping. Since the SHE version of this scheme is already visited in the previous section, therefore, we directly discuss the key switching (dimension reduction) and modulus switching which is as follows:

Key Switching:

It consists of two basic operations: $\mathsf{BitDecomp}$ and $\mathsf{PowersofTwo}$ as follows:

- $\mathsf{BitDecomp}(\boldsymbol{x} \in R_q^n, q)$ decomposes $\boldsymbol{x}$ into its bit representations $\boldsymbol{u} \in R_2^{n \cdot \lceil \log q \rceil}$. We do this by first writing $\boldsymbol{x} = \sum_{i=0}^{\lceil \log q \rceil} 2^i \cdot \boldsymbol{u}_i \in R_2^n$ then output $\mathbf{u} = (\boldsymbol{u}_0, \boldsymbol{u}_1, \cdots, \boldsymbol{u}_{\lceil \log q \rceil}) \in R_2^{n \cdot \lceil \log q \rceil}$.
- $\mathsf{PowersofTwo}(\boldsymbol{x} \in R_q^n, q)$ expands $\boldsymbol{x}$ into $\boldsymbol{u} \in R_2^{n \cdot \lceil \log q \rceil}$ that has copies of $\boldsymbol{x}$ multiplied by powers of 2. The output is $(\boldsymbol{x}, 2\boldsymbol{x}, \cdots, 2^{\lceil \log q \rceil}\boldsymbol{x}) \in R_2^{n \cdot \lceil \log q \rceil}$.

Lemma 5. $\langle \mathsf{BitDecomp}(c, q), \mathsf{PowersofTwo}(s, q) \rangle = \langle c, s \rangle \mod q$.

Proof. Using the definition of $\mathsf{BitDecomp}$ and $\mathsf{PowersofTwo}$ from above, we have

$$\begin{aligned}
\langle \mathsf{BitDecomp}(c, q), \mathsf{PowersofTwo}(s, q) \rangle &= \sum_{i=0}^{\lceil \log q \rceil} \langle c_i, 2^i \cdot \boldsymbol{s} \rangle = \sum_{i=0}^{\lceil \log q \rceil} 2^i \cdot \langle c_i, \boldsymbol{s} \rangle \\
&= \sum_{i=0}^{\lceil \log q \rceil} \langle 2^i \cdot c_i, \boldsymbol{s} \rangle = \left\langle \sum_{i=0}^{\lceil \log q \rceil} 2^i \cdot c_i, \boldsymbol{s} \right\rangle \\
&= \langle c, s \rangle \mod q.
\end{aligned}$$

□

In brief, the key switching technique can be defined by the following two operations:

- SwitchKeyGen($s_1 \in R_q^{n_1}, s_2 \in R_q^{n_2}$):
  1. Generate a public key A as defined in scheme but with secret key s_2 and parameter $n = n_1 \cdot \lceil \log q \rceil$.
  2. Set a matrix $B = [\mathsf{PowersofTwo}(s_1) | O]$, as a first column containing $\mathsf{PowersofTwo}(s_1)$ and padd some zeros column to match the size of A.
  3. Set $C = A + B$, and output $\tau_{s_1 \to s_2} = C$.
- SwitchKey($\tau_{s_1 \to s_2}, c_1$): Output $c_2 = \mathsf{BitDecomp}(c_1)^T \cdot C$.

The following lemma shows that the key switching works well.

Lemma 6. *Let s_1, s_2, q, A, B, C, be as in $SwitchKeyGen(s_1, s_2)$, and let $A \cdot s_2 = 2e_2 \in R_q^N$. Let $c_1 \in R_q^{n_1}$ and $c_2 \leftarrow SwitchKeyGen(\tau_{s_1 \to s_2}, c_1)$. Then we have*

$$\langle c_2, s_2 \rangle = 2\langle \mathsf{BitDecomp}(c_1), e_2 \rangle + \langle c_1, s_1 \rangle \mod q.$$

Proof. From the definition, we have

$$\begin{aligned}
\langle c_2, s_2 \rangle =& \langle \mathsf{BitDecomp}(c_1)^T \cdot C, s_2 \rangle \\
=& \mathsf{BitDecomp}(c_1)^T \cdot C \cdot s_2 \\
=& \mathsf{BitDecomp}(c_1)^T \cdot (A + B) \cdot s_2 \\
=& \langle \mathsf{BitDecomp}(c_1)^T \cdot (2e_2 + \mathsf{PowersofTwo}(s_1)) \\
=& 2\langle \mathsf{BitDecomp}(c_1), e_2 \rangle + \langle \mathsf{BitDecomp}(c_1), \mathsf{PowersofTwo}(s_1) \rangle \\
=& 2\langle \mathsf{BitDecomp}(c_1), e_2 \rangle + \langle c_1, s_1 \rangle \mod q.
\end{aligned}$$

□

Modulus Switching:

The authors present the modulus switching changing the ciphertext $c_1 \in R_q^2$ to $c_2 \in R_p^2$ encrypting the same message in this paper is a variant of the one used in [12]. Suppose we have a ciphertext $\mathbf{c} = (c_1, c_2) \in R_q^2$, and consider c_2 to be the vector closest(using ℓ_1 norm) to $(p/q) \cdot c_1$ such that $c_2 \equiv c_1 \mod 2$. The modulus switching technique can be explained by the following lemma:

Lemma 7. *Let d be the degree of the ring and $q > p > r$ be positive integers satisfying $q \equiv p \equiv 1 \mod r$. Let $c_1 \in R^n$ and $c_2 \leftarrow Scale(c, q, p, r)$. Then, for any $s \in R^n$ with $\|[\langle c_1, s \rangle]_q\| < q/2 - (q/p) \cdot (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot \ell_1^{(R)}(s)$, we have*

$$[\langle c_2, s \rangle]_p = [\langle c_1, s \rangle]_q \mod r \text{ and } \|[\langle c_2, s \rangle]_p\| < (p/q) \cdot \|[\langle c_1, s \rangle]_q\| + (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot \ell_1^{(R)}(s).$$

Proof. We have $[\langle c_1, s\rangle]_q = \langle c_1, s\rangle - kq$ for some k and for the same k, let $e_p = \langle c_2, s\rangle - kp \in R$. Then

$$\begin{aligned}
\|e_p\| &= \| - kp + (p/q) \cdot c_1, s\rangle + \langle c_2 - (p/q)\cdot, s\rangle\| \\
&\leq \| - kp + (p/q) \cdot c_1, s\rangle\| + \|\langle c_2 - (p/q)\cdot, s\rangle\| \\
&\leq (p/q) \cdot \|[\langle c_1, s\rangle]_q\| + \gamma(R) \cdot \sum_{j=1}^{n} \|c_2[j] - (p/q) \cdot c[j]\| \cdot \|s[j]\| \\
&\leq (p/q) \cdot \|[\langle c_1, s\rangle]_q\| + \gamma(R) \cdot (r/2) \cdot \sqrt{d} \cdot \ell_1^{(R)}(s) \\
&< p/2
\end{aligned}$$

and modulo r, we have $[\langle c_2, s\rangle]_p = e_p = \langle c_2, s\rangle - kp = \langle c_1, s\rangle - kq = [\langle c - 1, s\rangle]_q$. □

2.3.4 Brakerski's Scheme

The Brakerski's Scheme [9] is a scale-invariant fully homomorphic encryption scheme depending only on the ratio between the modulus q and the initial noise level B relying on Regev's basic public-key encryption scheme. In this work, the noise after the multiplication (without any refresh) grows linearly ($B \to B \cdot \text{poly}(n)$) while in all previous works noise grows quadratically ($B \to B^2 \cdot \text{poly}(n)$). The advantage of the scheme is that it uses same modulus throughout the evaluation process so it does not need modulus switching like other's scheme.

Encryption Scheme

For a given security parameter n, let $q(n)$ be an integer and $\chi = \chi(n)$ be a distribution over $\mathbb{Z}$. Set a secret key $\boldsymbol{s} = (s[1], \cdots, s[n]) \in \mathbb{Z}_p^n$. Let $N = (n+1)\cdot(\log q + O(1))$. Sample $A \leftarrow \mathbb{Z}^{N\times n}$ and $e \leftarrow \chi^N$. Compute $b = [A\cdot\boldsymbol{s}+e]_q$. Set a public key $P = [b|-A] \in \mathbb{Z}^{N\times(n+1)}$. Encrypt a message $m \in \{0,1\}$ as:

- Select a random $\boldsymbol{r} \in \{0,1\}^N$.
- Set $\tilde{m} = (m, 0, \cdots, 0) \in \{0,1\}^{n+1}$.
- Output the ciphertext $c = [P^T \cdot r + \lfloor q/2 \rfloor \cdot \tilde{m}]_q \in \mathbb{Z}_q^N$.

To recover the message m do the following:

- Compute $c_0 = [\langle c, (1, s)\rangle]_q$.
- Output $\tilde{m} = [\lfloor 2 \cdot c_0/q \rfloor]_2$.

Homomorphic Properties

Let the encryption of $m \in \{0, 1\}$ be a vector $c \in \mathbb{Z}_q^n$ as in Regev's public key encryption scheme such that $[\langle c, s\rangle]_q = \lfloor q/2 \rfloor \cdot m + e$ with $|e| \leq E$. Consider the invariant case, and set $c' = c/q$. Then $[\langle c', s\rangle]_1 = 1/2 \cdot m + e'$ with $|e'| \leq E/q = \epsilon$. Homomorphic addition can be defined as follows:

$$c_{\text{add}} = c_1 \oplus c_2$$

encrypting $[m_1 + m_2]_2$, with noise approximately 2ϵ. Homomorphic multiplication is defined as follows:

$$c_{\text{mult}} = 2 \cdot c_1 \otimes c_2$$

and can be decrypted using tensored secret key $s \otimes s$ because

$$\langle 2 \cdot c_1 \otimes c_2, s \otimes s\rangle = 2\langle c_1, s\rangle \cdot \langle c_2, s\rangle.$$

Now we show $[2\langle c_1, s\rangle \cdot \langle c_2, s\rangle]_1 = 1/2m_1m_2 + e'$, for small e' to see the definition works well. Let $I_1, I_2 \in \mathbb{Z}$ and rational numbers $|e_1|, |e_2| < \epsilon$ such that

$$\begin{aligned}\langle c_1, s\rangle &= 1/2m_1 + e_1 + I_1\\ \langle c_2, s\rangle &= 1/2m_2 + e_2 + I_2.\end{aligned}$$

Then,

$$\begin{aligned}2\langle c_1, s\rangle \cdot \langle c_2, s\rangle &= 2 \cdot (1/2m_1 + e_1 + I_1) \cdot (1/2m_2 + e_2 + I_2)\\ &= 1/2m_1m_2 + 2(e_1I_2 + e_2I_1) + (e_1m_2 + e_2m_1 + 2e_1e_2)\\ &\quad + (m_1I_2 + m_2I_1 + 2I_1I_2).\end{aligned}$$

As we can see that $m_1 I_2 + m_2 I_1 + 2 I_1 I_2 \in \mathbb{Z}$ and also the term $2e_1e_2$ that squares the noise in [12] and [10] can be ignored as $|2e_1e_2| \leq 2\epsilon^2 \ll \epsilon$. Moreover, using the triangular inequality we have $|e_1 m_2 + e_2 m_1| \leq 2\epsilon$. Brakerski in [9] showed the significant noise $2(e_1 I_2 + e_2 I_1)$ is bounded by $O(\|s\|_1) \cdot \epsilon$ by choosing $q \leq 2^n$ and $\|s\|_1 \approx n \cdot q$ is independent of b and q and only depends on n. Hence we have

$$\begin{aligned}
[2\langle c_1, s\rangle \cdot \langle c_2, s\rangle]_1 =& 1/2 m_1 m_2 + 2(e_1 I_2 + e_2 I_1) + (e_1 m_2 + e_2 m_1 + 2e_1 e_2) \\
\approx& 1/2 m_1 m_2 + 2(e_1 I_2 + e_2 I_1) \\
\approx& 1/2 m_1 m_2 + e',
\end{aligned}$$

where $e' = 2(e_1 I_2 + e_2 I_1)$.

Vector Decomposition and Key Switching

Vector decomposition is same as in [10] to reduce the norm of s. Initially, $\|s\|_1 \leq n \cdot q$, as the elements of the secret key are sampled from $(-q/2, q/2]$. Vector decomposition uses two operations BitDecomp and PowersofTwo same as in the BGV scheme [10].

Chapter 3

Secure Matrix Computations on Encrypted Data

In this chapter, we show how to pack large messages such as vectors, matrices into a single polynomial to make encryption scheme more efficient provided scheme supports polynomial operations homomorphically.

3.1 Previous Works

In this section, we briefly review the previous packing methods proposed for homomorphic matrix operations.

3.1.1 Packing Method for Large Integers

Lauter et al. [62] introduced a method to pack a large integer into a single ciphertext, and it enables efficient computation of sums and products over packed ciphertexts. Specially, a message M of up to n bits is broken into a binary vector $(m_0, \cdots, m_{n-1})$ and associated with a polynomial

$$\mathrm{pm}(M) = \sum_{i=0}^{n-1} m_i x^i,$$

of degree less than or equal to $n-1$ and finally encrypts M as $\mathrm{ct}(M) := \mathrm{Enc}(\mathrm{pm}(M), \mathrm{pk})$. Note that $\mathrm{pm}(M)|_{x=2} = M$. Moreover, for two given plaintexts M and M', the homo-

morphic addition $\mathrm{ct}(M) \oplus \mathrm{ct}(M')$ gives the polynomial addition $\mathrm{pm}(M) + \mathrm{pm}(M')$ on encrypted data under the correctness [62, Lemma 3.3]. However, the integer multiplication causes a problem as polynomial multiplication $\mathrm{pm}(M) \cdot \mathrm{pm}(M')$ has degree larger than n. Hence, their method can only encodes integers of at most (n/d)-bit when it requires d homomorphic multiplications. Their method is effective in computing low degree multiplications.

3.1.2 Packing Method for Secure Inner Product

Yasuda et al. [74] proposed two types of packed ciphertexts as follows: for an integral vector $A = (a_0, \cdots, a_{m-1})$ of length $m \leq n$, set

$$\mathrm{pm}^{(1)}(A) = \sum_{i=0}^{m-1} a_i x^i \text{ and } \mathrm{pm}^{(2)}(A) = -\sum_{i=0}^{m-1} a_i x^{n-i}, \tag{3.1}$$

and then packed ciphertexts are defined as $\mathrm{ct}^{(i)}(A) := \mathrm{Enc}(\mathrm{pm}^{(i)}(A), \mathrm{pk})$ for $i = 1, 2$. The first type is same as Lauter et al.'s method [62], but the second one enables efficient secure computation of inner product. Specially, for two vectors A and B of the same length $m \leq n$, only one homomorphic multiplication over $\mathrm{ct}^{(1)}(A)$ and $\mathrm{ct}^{(2)}(B)$ can give the inner product $\langle A, B \rangle$ on encrypted data; see [74, Proposition 1]. This computation is useful for secure distance computations such as the Euclidean and the Hamming distances; see [74] for more details.

3.1.3 Modified Packing Method for Large Integer Entries

When handling with small integers (e.g., 1 or 2 bits), the packing method in [74] is effective and sufficient for secure statistics. However, for integers of practical bit-size (e.g., even for 16 or 32 bit), the packing method enforces to set the parameter t of the plaintext space R_t to be considerably large, and it then causes slow performance of the SHE scheme. Yasuda et al. [75] modified the packing method in [74] for large-size integers as follows.

Given an integer m, let $A = (a_0, \cdots, a_{m-1})$ be a vector of length m with entries a_i less than p bits. For a chosen integer $r > 0$ (e.g., $r = 2$), we write each integral entry a_i

in the base-r representation, namely

$$a_i = \sum_{u=0}^{d-1} a_{i,u} r^u \text{ with } a_{i,u} \in \{0, 1, \cdots, r-1\},$$

where $d = \lceil \log_r 2^p \rceil$. We associate to A the following two polynomials:

$$\mathrm{pm}^{(1)}_{m,p,r}(A) = \sum_{i=0}^{m-1} \left(\sum_{u=0}^{d-1} a_{i,u} x^u \right) x^{2id}, \tag{3.2}$$

$$\mathrm{pm}^{(2)}_{m,p,r}(A) = -\sum_{i=0}^{m-1} \left(\sum_{u=0}^{d-1} a_{i,u} x^u \right) x^{n-2id}, \tag{3.3}$$

and then packed ciphertexts are defined as $\mathrm{ct}^{(i)}_{m,p,r}(A) := \mathrm{Enc}(\mathrm{pm}^{(i)}_{m,p,r}(A), \mathrm{pk})$ for $i = 1, 2$. Then we have the following theorem [75, Theorem 1] on the secure inner product between A and B with large entries.

Theorem 8. *Let $A = (a_0, \cdots, a_{m-1})$ and $B = (b_0, \cdots, b_{m-1})$ be two integral vectors of same length m with entries less than p bits. Assume that $n \geq 2md$ and $t > m(r-1)^2 d$, where $d = \lceil \log_r 2^p \rceil$ for a fixed positive integer r. Let*

$$\mathsf{ct} = \mathsf{ct}^{(1)}_{m,p,r}(A) * \mathsf{ct}^{(2)}_{m,p,r}(B),$$

and let $Dec(\mathsf{ct}, \mathsf{pk}) = \sum_{i=0}^{n-1} m_i x^i \in R_t$ denote the decryption result. Then under the condition of Lemma 4 for the ciphertext ct, the sum $\sum_{i=0}^{2d-1} m_i r^i \in \mathbb{Z}$ gives the inner product $\langle A, B \rangle$.

3.1.4 Wang et al.'s Packing Method for Secure Matrix Multiplication

In this section, we review Wang et al.'s [69] method for secure matrix multiplication. This method generalise the technique for secure inner product presented in [70]. Their packing method is also performed over the BV scheme. Therefore we only state the packing technique of this scheme. Denote variant vector $X = (1, x, \cdots, x^{m-1})^T$, coefficient vector $\mathrm{Vec}(a) = (a_0, \cdots, a_{m-1})$ for $a = a_0 + a_1 x + \cdots + a_{m-1} x^{m-1}$. Let $A_i^{(r)}$ and $B_j^{(c)}$ be i-th

and j-th column of the matrix $\mathbf{A}$ of the size $m \times m$. Now, pack matrix $\mathbf{A}$ as follows:

$$\mathbf{A}(x) = \sum_{i=1}^{m} x^{(i-1)m} A_i^{(r)} X + \sum_{j=1}^{m} x^{2jm^2-m} B_j^{(c)} X.$$

Theorem 9 (Theorem 2, from [69]). *For two given matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of the size $m \times m$, the multiplication $\boldsymbol{AB} = (\langle A_i^{(r)}, B_j^{(c)} \rangle)_{m \times m}$ can be obtained by computing coefficients vector inner products of shifted $\boldsymbol{A}(x)$ and $\boldsymbol{B}(x)$ when $n \geq 4m^3$. In details*

$$\langle Vec(x^{2jm^2-m} \boldsymbol{A}(x)), Vec(x^{(i-1)m} \boldsymbol{B}(x) \rangle = \langle A_i^{(r)}, B_j^{(c)} \rangle.$$

3.1.5 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 [45, 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 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 [47] can be used to make rotations more efficient

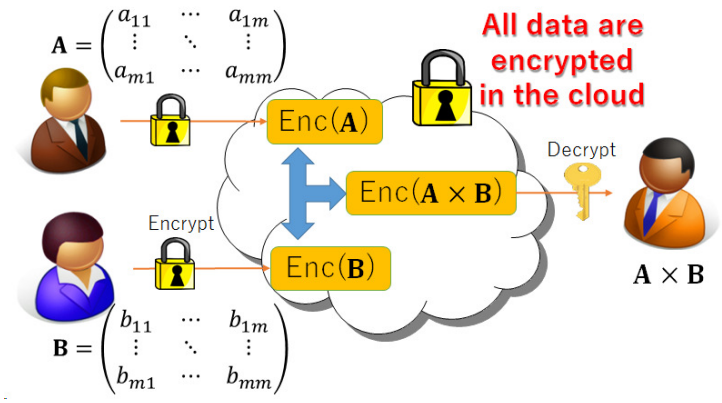

Fig. 3.1 An image of secure matrix multiplication $\mathbf{A} \times \mathbf{B}$ in the cloud

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. [57] slightly modified the matrix-vector multiplication of [45, Section 4.3] for homomorphic evaluation of principal component analysis and linear regression.)

3.1.6 State-of-the-Art

Recently, Jiang et al. [50] proposed a new method for matrix multiplication which focuses on using only homomorphic encryption and evaluated a neural network on the MNIST data set [72] using this technique. They also considered packing an entire image into a single ciphertext compared to the approach of Dowlin et al. [27] who put only one pixel per ciphertext but evaluated large batches of images at a time. They achieved good performance, evaluating 64 images in slightly under 24 seconds but with worse amortized performance.

As another work, Juvekar et al. presented GAZELLE [51] that deploys automorphism structure of an underlying homomorphic encryption scheme to perform matrix-vector multiplication, thereby improving the performance significantly. It took 30 milliseconds to classify one image from the MNIST data set and has an online bandwidth cost of 0.5 MB. Even though these fixed protocols achieve relatively fast run-time, they require interaction between protocol participants, resulting in high bandwidth usage.

3.2 Our Secure Matrix Multiplications

In this section, we propose our packing methods for secure matrix multiplications over the ring-based homomorphic encryption scheme.

3.2.1 Single Matrix Multiplication

In this section, we propose packing methods to efficiently compute secure matrix multiplication. Notice that by using the packing methods of Yasuda et al. [74, 75] described in the previous section, one needs m^2 secure inner product computations (i.e., m^2 homomorphic multiplications) to compute a matrix multiplication of two $m \times m$ matrices. In this section, for each of the packing method proposed by Yasuda et al., we propose two methods to reduce the number of computations: the first one requires m *homomorphic multiplications* and the second one requires *only one homomorphic multiplication*.

Binary Entries

Let $\mathbf{A}$ be an $m \times m$ matrix with binary entries. Let $A_1, \cdots, A_m$ denote the row vectors of $\mathbf{A}$ and $A_1^T, \cdots, A_m^T$ the column vectors of $\mathbf{A}$. In order to compute the matrix multiplication $\mathbf{AB}$ of two binary matrices $\mathbf{A}$ and $\mathbf{B}$, one needs to compute the inner products $\langle A_i, B_j^T \rangle$ for $i, j = 1, \cdots, m$. We pack each row $A_i = (a_{i,0}, \cdots, a_{i,m-1})$ and each column $A_j^T = (a_{j,0}, \cdots, a_{j,m-1})$ of $\mathbf{A}$ as in (3.1), namely:

$$\mathrm{pm}^{(1)}(A_i) := \sum_{u=0}^{m-1} a_{i,u} x^u, \ \mathrm{pm}^{(2)}(A_j^T) := -\sum_{v=0}^{m-1} a_{j,v} x^{n-v}. \tag{3.4}$$

It follows from Section 3.1.2 that the constant term of $\mathrm{pm}^{(1)}(A_i) \times \mathrm{pm}^{(2)}(B_j^T)$ is the inner product $\langle A_i, B_j^T \rangle$ (note that "$\times$" denotes the multiplication in the ring $R = \mathbb{Z}[x]/(x^n+1)$).

We define the following two types of polynomials in R associated to a given matrix $\mathbf{A}$:

$$\begin{aligned}\mathrm{Pol}^{(1)}(\mathbf{A}) :=& \mathrm{pm}^{(1)}(A_1) + \cdots + \mathrm{pm}^{(1)}(A_m)x^{m(m-1)} \\ =& \sum_{i=1}^{m} \mathrm{pm}^{(1)}(A_i)x^{(i-1)m}, \\ \mathrm{Pol}^{(2)}(\mathbf{A}) :=& \mathrm{pm}^{(2)}(A_1^T) + \cdots + \mathrm{pm}^{(2)}(A_m^T)x^{m^2(m-1)} \\ =& \sum_{j=1}^{m} \mathrm{pm}^{(2)}(A_j^T)x^{(j-1)m^2}.\end{aligned}$$

Define the packed ciphertexts for a given matrix $\mathbf{A}$ to be

$$\mathrm{ct}_{\mathrm{mat}}^{(\mathbf{i})}(\mathbf{A}) := \mathrm{Enc}(\mathrm{Pol}^{(\mathbf{i})}(\mathbf{A}), \mathrm{pk}), \text{ for } i = 1, 2.$$

In order to get the matrix multiplication $\mathbf{AB}$, we need to obtain the inner products $\langle A_i, B_j^T \rangle$ for $i, j = 1, \cdots, m$. Here we present our two approaches.

Theorem 10 (First Packing Method)**.** *Assume that $n \geq m^2$. For each $j = 1, \cdots, m$, let*

$$\mathsf{ct}_j = \mathrm{ct}_{\mathrm{mat}}^{(1)}(\mathbf{A}) * \mathrm{ct}^{(2)}(\mathbf{B_j^T})$$

and let $Dec(\mathsf{ct}_j, \mathsf{sk}) \in R_t$ denote the decryption result. Then under the condition of Lemma 4 for the ciphertext ct_j, for each $j = 1, \cdots, m$, the inner product $\langle A_i, B_j^T \rangle$ is the coefficient of $x^{(i-1)m}$ in $Dec(\mathsf{ct}_j, \mathsf{sk})$.

Proof. It follows from Section 3.1.2 (and clear from definition) that

$$\mathrm{pm}^{(1)}(A_i) \times \mathrm{pm}^{(2)}(B_j^T) = \langle A_i, B_j^T \rangle + \text{other terms of degree } (n - v + u),$$

with $u \neq v$ and $u, v \in \{0, \cdots, m-1\}$, and so the constant term of $\mathrm{pm}^{(1)}(A_i) \times \mathrm{pm}^{(2)}(B_j^T) \in R$ gives us the inner product of A_i and B_j^T. Now one has that

$$\begin{aligned}\mathrm{Dec}(\mathsf{ct}_j, \mathsf{sk}) &= \mathrm{Pol}^{(1)}(\mathbf{A}) \times \mathrm{pm}^{(2)}(B_j^T) \\ &= \sum_{i=1}^{m} \mathrm{pm}^{(1)}(A_i) \times \mathrm{pm}^{(2)}(B_j^T)x^{(i-1)m}.\end{aligned}$$

For a fixed index i, we have

$$\begin{aligned}\mathrm{pm}^{(1)}(A_i) \times \mathrm{pm}^{(2)}(B_j^T)x^{(i-1)m} &= \langle A_i, B_j^T\rangle x^{(i-1)m} \\ &+ \ \tilde{O}(n-v+u+(i-1)m),\end{aligned}$$

with $u \neq v$ and $u, v \in \{0, \cdots, m-1\}$. We write $\tilde{O}$ for "other terms of degree" due to space limitation. Since the exponents of x is modulo n and therefore $n-v+u+(k-1)m$ is never equal to $(i-1)m$ for $u \neq v$ and $i, k \in \{1, \cdots, m\}$. This implies that the terms of degree $(i-1)m$ in $\mathrm{Pol}^{(1)}(\mathbf{A}) \times \mathrm{pm}^{(2)}(B_j^T)$ is exactly the term of degree $(i-1)m$ in $\mathrm{pm}^{(1)}(A_i) \times \mathrm{pm}^{(2)}(B_j^T)x^{(i-1)m}$. Hence the inner product $\langle A_i, B_j^T\rangle$ is the coefficient of $x^{(i-1)m}$ in $\mathrm{Dec}(\mathsf{ct}_j, \mathsf{sk})$. □

Remark 11. *With this approach, after m homomorphic multiplications over the SHE scheme, we get the resulting matrix multiplication* $\mathbf{AB}$.

Theorem 12 (Second packing method)**.** *Assume that $n \geq m^3$. For each $j = 1, \cdots, m$, let*

$$\mathsf{ct} = \mathrm{ct}_{\mathrm{mat}}^{(1)}(\mathbf{A}) * \mathrm{ct}_{\mathrm{mat}}^{(2)}(\mathbf{B})$$

and let $Dec(\mathsf{ct}, \mathsf{sk}) \in R_t$ denote the decryption result. Then under the condition of Lemma 4 for the ciphertext ct, for each i and j, the inner product $\langle A_i, B_j^T\rangle$ is the coefficient of $x^{(j-1)m^2+(i-1)m}$ in $Dec(\mathsf{ct}, \mathsf{sk})$.

Proof. The proof is similar to that of Theorem 10. Over the ring R, we have

$$\begin{aligned}\mathrm{Dec}(\mathsf{ct}, \mathsf{sk}) &= \mathrm{Pol}^{(1)}(\mathbf{A}) \times \mathrm{Pol}^{(2)}(\mathbf{B}) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} \mathrm{pm}^{(1)}(A_i) \times \mathrm{pm}^{(2)}(B_j^T)x^{(j-1)m^2+(i-1)m}.\end{aligned}$$

As in the proof of Theorem 10, the term of degree $(j-1)m^2+(i-1)m$ in $\mathrm{Dec}(\mathsf{ct}, \mathsf{sk})$ is exactly the term of degree $(j-1)m^2+(i-1)m$ in $\mathrm{pm}^{(1)}(A_i) \times \mathrm{pm}^{(2)}(B_j^T)x^{(j-1)m^2+(i-1)m} \in R$. Hence the coefficient of $x^{(j-1)m^2+(i-1)m}$ in $\mathrm{Dec}(\mathsf{ct}, \mathsf{sk})$ gives the inner product $\langle A_i, B_j^T\rangle$. □

Remark 13. *In this approach, we need to do only one homomorphic multiplication on packed ciphertexts for secure matrix multiplication.*

Non-Binary Eentries

In this subsection, we adapt the method in Yasuda et al. [75] described in Section 3.1.3 to propose packing methods for multiplying integral matrices with non-binary entries. Let $\mathbf{A}$ and $\mathbf{B}$ be two $m \times m$ matrices whose entries are integers of less than p bits. Let $A^{(1)}, \cdots, A^{(m)}$ and $B^{(1)}, \cdots, B^{(m)}$ be the rows of $\mathbf{A}$ and $\mathbf{B}^T$ respectively. For $i = 0, \cdots, m-1$, we write $A^{(i)} = (a_0^{(i)}, \cdots, a_{m-1}^{(i)})$ and $B^{(i)} = (b_0^{(i)}, \cdots, b_{m-1}^{(i)})$. For a chosen integer $r > 0$, we write each integral entry $a_k^{(i)}$ in the base-r representation, namely

$$a_k^{(i)} = \sum_{u=0}^{d-1} a_{ku}^{(i)} r^u \text{ with } a_{ku}^{(i)} \in \{0, 1, \cdots, r-1\},$$

where $d = \lceil \log_r 2^p \rceil$, as in Section 3.1.3 (we took $r = 2$ in our experiments, and then $d = p$). We pack $a_k^{(i)}$ as

$$a_k^{(i)}(x) := \sum_{u=0}^{d-1} a_{ku}^{(i)} x^u \in R = \mathbb{Z}[x]/(x^n + 1).$$

We associate to each row $A^{(i)}$ and column $B^{(j)}$ of $\mathbf{A}$ and $\mathbf{B}$ respectively the following polynomials in the ring R:

$$\mathrm{pm}^{(1)}_{m,p,r}(A^{(i)}) = \sum_{k=0}^{m-1} a_k^{(i)}(x) x^{2kd}, \tag{3.5}$$

$$\mathrm{pm}^{(2)}_{m,p,r}(B^{(j)}) = -\sum_{l=0}^{m-1} b_l^{(j)}(x) x^{n-2ld}, \tag{3.6}$$

We define the following polynomials in R associated to $\mathbf{A}$ and $\mathbf{B}$:

$$\begin{aligned}
\mathrm{Pol}^{(1)}(\mathbf{A}) &:= \mathrm{pm}^{(1)}_{m,p,r}(A^{(1)}) + \cdots + \mathrm{pm}^{(1)}_{m,p,r}(A^{(m)}) x^{(m-1)2md} \\
&= \sum_{i=1}^{m} \mathrm{pm}^{(1)}_{m,p,r}(A^{(i)}) x^{(i-1)2md}, \\
\mathrm{Pol}^{(2)}(\mathbf{B}) &:= \mathrm{pm}^{(2)}_{m,p,r}(B^{(1)}) + \cdots + \mathrm{pm}^{(2)}_{m,p,r}(B^{(m)}) x^{(m-1)2m^2 d} \\
&= \sum_{j=1}^{m} \mathrm{pm}^{(2)}_{m,p,r}(B^{(j)}) x^{(j-1)2m^2 d}.
\end{aligned}$$

Define the packed ciphertexts for a given matrix $\mathbf{A}$ to be

$$\mathrm{ct}^{(\mathbf{i})}_{\mathrm{mat}}(\mathbf{A}) := \mathrm{Enc}(\mathrm{Pol}^{(\mathbf{i})}(\mathbf{A}), \mathrm{pk}), \text{ for } i = 1, 2.$$

As in the case of binary matrix multiplication, we here present two approaches.

Theorem 14 (First packing method)**.** *Assume that $n \geq 2md(m+1)$. For each $j = 1, \cdots, m$, let*

$$\mathsf{ct}_j = \mathsf{ct}_{\mathrm{mat}}(\mathbf{A}) * \mathrm{ct}^{(2)}_{m,p,r}(B^{(j)}),$$

and let $Dec(\mathsf{ct}_j, \mathsf{sk}) \in R_t$ denote the decryption result. Then under the condition of Lemma 4 for the ciphertext ct_j, for each $j = 1, \cdots, m$, the inner product $\langle A^{(i)}, B^{(j)} \rangle$ is the sum of the terms of degree greater or equal to $(i-1)2md$ and less than $(i-1)2md+2d$ in $Dec(\mathsf{ct}_j, \mathsf{sk})$ evaluated at $x = r$.

Proof. Fix i and j. Over the ring R, we have that

$$\begin{aligned}\mathrm{pm}^{(1)}_{m,p,r}(A^{(i)}) \times \mathrm{pm}^{(2)}_{m,p,r}(B^{(j)}) = & \underbrace{\sum_{k=0}^{m-1} a^{(i)}_k(x) b^{(j)}_k(x)}_{I(i)} \\ & + \underbrace{\left(- \sum_{k=0}^{m-1} \sum_{l=0}^{k-1} a^{(i)}_k(x) b^{(j)}_l(x) x^{n+(k-l)2d} \right)}_{II(i)} \\ & + \underbrace{\left(- \sum_{k=0}^{m-1} \sum_{l=k+1}^{m-1} a^{(i)}_k(x) b^{(j)}_l x^{n+(k-l)2d} \right)}_{III(i)}.\end{aligned}$$

One gets that the degrees of terms in $I(i), II(i)$ and $III(i)$ are in the intervals $[0, 2d-2]$, $[2d, 2md-2]$ and $[n-(m-1)2d, n-2]$ respectively. It follows that the sum of terms of degree less than $2d$ in $\mathrm{pm}^{(1)}_{m,p,r}(A^{(i)}) \times \mathrm{pm}^{(2)}_{m,p,r}(B^{(j)}) \in R$ evaluated at $x = r$ gives us the inner product $\langle A^{(i)}, B^{(j)} \rangle$. Now for each $j = 1, \cdots, m$

$$\begin{aligned}\mathrm{Dec}(\mathsf{ct}_j, \mathsf{sk}) &= \mathrm{Pol}^{(1)}(\mathbf{A}) \times \mathrm{pm}^{(2)}_{m,p,r}(B^{(j)}) \\ &= \sum_{i=1}^{m} \mathrm{pm}^{(1)}_{m,p,r}(A^{(i)}) \times \mathrm{pm}^{(2)}_{m,p,r}(B^{(j)}) x^{(i-1)2md}.\end{aligned}$$

It then follows that for every $i = 1, \cdots, m$ the inner product $\langle A^{(i)}, B^{(j)} \rangle$ is the sum of terms of degree greater than or equal to $(i-1)2md$ and less than $(i-1)2md + 2d$ in $\mathrm{Dec}(\mathsf{ct}_j, \mathsf{sk})$ evaluated at $x = r$. □

Theorem 15 (Second Packing Method)**.** *Assume that* $n \geq 2m^3d + 2md + 2d$. *Let*

$$\mathsf{ct} = \mathrm{ct}_{\mathrm{mat}}^{(1)}(\mathbf{A}) * \mathrm{ct}_{\mathrm{mat}}^{(2)}(\mathbf{B})$$

and let $Dec(\mathsf{ct}, \mathsf{sk}) \in R_t$ *denote the decryption result. Then under the condition of Lemma 4 for the ciphertext* ct*, for each* i *and* j*, the inner product* $\langle A^{(i)}, B^{(j)} \rangle$ *is the sum of terms of degree greater than or equal to* $(i-1)2md + (j-1)2m^2d$ *and less than* $(i-1)2md + (j-1)2m^2d + 2d$ *in* $Dec(\mathsf{ct}, \mathsf{sk})$ *evaluated at* $x = r$.

Proof. The proof follows easily from that of Theorems 12 and 14. □

3.2.2 Multiple Matrix Multiplications

Methods presented in the previous section are only for one time secure multiplication between two matrices. Moreover, those methods enforce us to take very large size of t or n for large entries (see Chapter 4 for details). In this section, we first extend our methods from just one matrix multiplication to multiple matrix multiplications, and then give some modifications for efficiency.

We present a packing method for secure multiple matrix multiplications with integer entries. Specifically, we generalize methods of [28] from single matrix multiplication to multiple matrix multiplications.

Case of Three Matrices

Let $\mathbf{A}$ be a matrix with size $m \times m$ whose entries are positive entries. As in the previous section, 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:

$$\begin{cases} \mathrm{pm}^{(1)}_{z,3}(A_i) := \sum_{u=0}^{m-1} a_{i,u} x^u, \\ \mathrm{pm}^{(2)}_{z,3}(A_i) := -\sum_{u=0}^{m-1} a_{i,u} x^{n-um^2-m+1}. \end{cases}$$

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

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

where $B^T_j = (b_{0,j}, \ldots, b_{m-1,j})$ and C^T_k are the j^{th} and the k^{th} columns of $\mathbf{B}$ and $\mathbf{C}$, respectively, and $\overline{B}^T_j = (b_{m-1,j}, \ldots, b_{0,j})$ (i.e., flipping the column B^T_j). Define three types of packed ciphertext for a matrix $\mathbf{A}$ to be

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

Theorem 16. *Assume $n \geq m^4$. Let*

$$\mathsf{ct}_{z,3} = \mathsf{ct}^{(1)}_{z,3}(\mathbf{A}) * \mathsf{ct}^{(2)}_{z,3}(\mathbf{B}) * \mathsf{ct}^{(3)}_{z,3}(\mathbf{C}),$$

and let $\mathrm{Dec}(\mathsf{ct}_{z,3}, \mathsf{sk}) \in R_t$ denote its decryption result. Then under the condition of Lemma 4 for the ciphertext $\mathsf{ct}_{z,3}$, for each $i, k \in \{1, \ldots, m\}$, the $(i,k)^{\text{th}}$ entry of the matrix $\mathbf{A} \times \mathbf{B} \times \mathbf{C}$ is the coefficient of $x^{(i-1)m+(k-1)m^3}$ in $\mathrm{Dec}(\mathsf{ct}_{z,3}, \mathsf{sk})$.

Proof. It follows from our construction that

$$\begin{aligned} \mathrm{pm}^{(1)}_{z,3}(A_i) \times \mathrm{pm}^{(1)}_{z,3}\left(\overline{B}^T_j\right) &= \sum_{u=1}^{m}\sum_{v=1}^{m} a_{iu} b_{m-v+1,j} x^{u+v-2} \\ &= \left\langle A_i, B^T_j \right\rangle x^{m-1} + \tilde{O}(u+v-2), \end{aligned}$$

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 $\mathrm{pm}^{(1)}_{z,3}(A_i) \times \mathrm{pm}^{(1)}_{z,3}\left(\overline{B}^T_j\right) \in R$ gives the inner product $\left\langle A_i, B^T_j \right\rangle$. Furthermore we have

$$\mathrm{pm}^{(1)}_{z,3}(A_i) \times \mathrm{Pol}^{(2)}_{z,3}(\mathbf{B}) = \sum_{j=1}^{m} \left\langle A_i, B^T_j \right\rangle x^{(j-1)m^2+m-1} + \tilde{O}(u+v-2+(j-1)m^2).$$

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

$$\begin{aligned}
&\mathrm{pm}^{(1)}_{z,3}(A_i) \times \mathrm{Pol}^{(2)}_{z,3}(\mathbf{B}) \times \mathrm{pm}^{(2)}_{z,3}\left(C^T_k\right) \\
&= \sum_{j=1}^{m} \left\langle A_i, B^T_j \right\rangle x^{(j-1)m^2+m-1} \times \mathrm{pm}^{(2)}_{z,3}\left(C^T_k\right) + \tilde{O}\left(n+u+v-m-1+(j-1)m^2-(w-1)m^2\right) \\
&= -\sum_{j=1}^{m}\sum_{w=1}^{m} \left\langle A_i, B^T_j \right\rangle c_{k,w} x^{n+(j-1)m^2-(w-1)m^2} + \tilde{O}\left(n+u+v-m-1+(j-w)m^2\right) \\
&= \left\langle V_i, C^T_k \right\rangle + \tilde{O}\left(n+u+v-m-1+(j-w)m^2\right)
\end{aligned}$$

with $w \neq j$ for $w, j \in \{0, \ldots, m-1\}$, where set $V_i = \left(\left\langle A_i, B^T_j \right\rangle\right)_{j=1}^{m}$ for $1 \leq i \leq m$. Under the condition of Lemma 4 for the ciphertext ct, we have

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

Furthermore, for fixed indices i and k, we have

$$\begin{aligned}
&\mathrm{pm}^{(1)}_{z,3}(A_i) \times \mathrm{Pol}^{(2)}_{z,3}(\mathbf{B}) \times \mathrm{pm}^{(2)}_{z,3}\left(C^T_k\right) x^{\alpha(i,k,m)} \\
&= \left\langle V_i, C^T_k \right\rangle x^{\alpha(i,k,m)} + \tilde{O}\left(n+u+v-m-1+(j-w)m^2+\alpha(i,k,m)\right)
\end{aligned}$$

with $\alpha(i,k,m) = (i-1)m + (k-1)m^3$, $u + v \neq m+1$ 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 $n + u + v - m - 1 + (j-w)m^2 + \alpha(h,\ell,m)$ is never equal to $\alpha(i,k,m)$ 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 $\alpha(i,k,m)$ in $\mathrm{Pol}^{(1)}_{z,3}(\mathbf{A}) \times \mathrm{Pol}^{(2)}_{z,3}(\mathbf{B}) \times \mathrm{Pol}^{(3)}_{z,3}(\mathbf{C})$ is exactly the term of degree α_{ikm} in $\mathrm{pm}^{(1)}_{z,3}(A_i) \times \mathrm{Pol}^{(2)}_{z,3}(\mathbf{B}) \times \mathrm{pm}^{(2)}_{z,3}\left(C^T_k\right) x^{\alpha(i,k,m)}$. Hence the inner product $\left\langle V_i, C^T_k \right\rangle$, which is equal to the

$(i,k)^{th}$ entry of $\mathbf{A}\times\mathbf{B}\times\mathbf{C}$, can be recovered as the coefficient of $x^{\alpha_{ikm}}$ in $\mathrm{Dec}(\mathsf{ct}_{z,3},\mathsf{sk})\in R_t$. □

Case of $\ell \geq 4$

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

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

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

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

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

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

Similar to Theorem 16, we obtain the following result for secure multiple matrix multiplications over our packed ciphertexts:

Theorem 17. *Assume* $n \geq m^{\ell+1}$. *Let*

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

and let $\mathrm{Dec}(\mathsf{ct}_{z,\ell}, \mathsf{sk}) \in R_t$ *denote its decryption result. Then under the condition of Lemma 4 for the ciphertext* $\mathsf{ct}_{z,\ell}$, *for each* $i, j \in \{1, \ldots, m\}$, *the* $(i,j)^{\text{th}}$ *entry of the matrix* $\prod_{h=1}^{\ell} \mathbf{A}_h$ *is the coefficient of* $x^{(i-1)m+(j-1)m^\ell}$ *in* $\mathrm{Dec}(\mathsf{ct}_{z,\ell}, \mathsf{sk})$.

3.2.3 Some Modifications

For matrices with large entries, our packing method needs very large t and hence huge q for successful decryption. Such parameter setting makes the BV scheme very slow. On the other hand, the packing method in Subsection 3.2.1 enables to take small t. However, it forces to set very large n since it requires $n \geq 2m^3p + 2mp + 2p$ from Theorem 10 (in which $d = p$ when $r = 2$) for matrices with p-bit entries. Here we give two modifications to get rid of these two problems. Specifically, we use the CRT method and the block-matrix method to reduce the size of t and n, respectively.

As described in the previous sections, 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 a setting makes the scheme very slow. Here we give two modifications to relax the problem.

CRT Method

In order to deal with huge plaintext size t, 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 $\mathbf{F}_{t_i}[x]/(x^n + 1)$ as the plaintext space for every t_i. On the other hand, it requires e ciphertexts for encrypting a message but this problem can

be alleviated through multi-threading. Moreover, smaller plaintext moduli require less number of moduli for leveled ciphertext spaces.

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.

3.3 Experimental Details

In this section, we show our implementation details over the BV scheme [12] and the BGV scheme [10].

3.3.1 Experiments Over BV Scheme

In this subsection, we report our implementation results over the BV scheme [12]. Specifically, we implemented previous methods for inner product, our methods for secure single matrix multiplication with binary and non-binary entries and then we show performance of our packing method with some modifications for secure multiple multiplications among two, three, and four matrices with size $M \times M$ with p - bit entries for $M = 16, 32$ and $p = 16, 32$ (for simplicity, we assume that size of precision entries in real matrices is same as integral entries). In the next section, we describe how to select parameters of the BV scheme for each method.

Parameter Settings

Our selection of parameters is followed from [62, Section 3.2.2]. In this subsection, we describe how to select parameters (n, q, t, σ) of the SHE scheme.

Single Matrix Multiplication We divide our description for single matrix multiplication into two cases where the entries of matrices $\mathbf{A}$ and $\mathbf{B}$ are binary and non-binary.

Binary Matrix Multiplication Here, we assume that the entries of two $m \times m$ matrices $\mathbf{A}$ and $\mathbf{B}$ are binary. As in [62], we fix $\sigma = 8$. For the matrix size m, it is sufficient to take the plaintext modulus as $t = m + 1$ since every entry of $\mathbf{AB}$ is not greater than m. In our experiments, we take $m = 16$ and 32 for practical use, and hence we set $t = 17$ and 33, respectively. Let ct_1 and ct_2 denote the two ciphertexts obtained by packing a part of or whole $\mathbf{A}$ and $\mathbf{B}$, respectively. For example, in our first packing method, we have $\mathrm{ct}_1 = \mathrm{ct}^{(1)}_{\mathrm{mat}}(\mathbf{A})$ and $\mathrm{ct}_2 = \mathrm{ct}^{(2)}_{\mathrm{pack}}(B_j^T)$ for some $1 \le j \le m$. In every packing method, we need to multiply $\mathrm{ct} := \mathrm{ct}_1 * \mathrm{ct}_2$ over ciphertexts for secure matrix multiplication $\mathbf{AB}$. In order to avoid decryption failure of the ciphertext ct, it requires $\|\langle \mathrm{ct}, \mathbf{s}\rangle\|_\infty < \mathbf{q/2}$ by Lemma 4. Let U denote an upper bound of the ∞-norm size $\|\langle \mathrm{ct}', \mathbf{s}\rangle\|_\infty$ for any fresh ciphertexts $\mathrm{ct}' \in (R_q)^2$. We clearly have

$$\|\langle \mathrm{ct}, \mathbf{s}\rangle\|_\infty = \|\langle \mathrm{ct_1}, \mathbf{s}\rangle \cdot \langle \mathrm{ct_2}, \mathbf{s}\rangle\|_\infty \le \mathbf{nU^2}$$

by the well-known fact that $\|a \cdot b\|_\infty \le n\|a\|_\infty \|b\|_\infty$ for $a, b \in R = \mathbb{Z}[x]/(x^n+1)$. According to the experimental estimation of [62], we may take $U = 2t\sigma^2\sqrt{n}$ in practice. Therefore it suffices to take a prime q satisfying

$$2n(2t\sigma^2\sqrt{n})^2 = 8n^2t^2\sigma^4 \le q. \tag{3.7}$$

With respect to the degree parameter n, the previous method [74] requires $n \ge m$, our first method $n \ge m^2$, and our second method $n \ge m^3$. In particular, we take $n = (2^5)^3 = 32768$ in our second method for $m = 32 = 2^5$. In this case, we approximately have $q \ge 2^{3+30+10+12} = 2^{55}$ by (3.7) since

$t \approx 2^5$. Then we always fix one prime q of 60-bit for the binary case. Note that with such q, we can avoid decryption failure of ct in every method since our second method requires the largest n and q. Furthermore, since the ciphertext modulus q is less than 64-bit, we estimate that the performance over a 64-bit machine would not change when we use smaller q. We remark that for such q, we need $n \geq 2048$ for more than 80-bit security (see [62] for more details on the security).

Non-Binary Matrix Multiplication In this case, we assume that the entries of two $m \times m$ matrices $\mathbf{A}$ and $\mathbf{B}$ are less than p-bit. In our experiments, we took $r = 2$ and hence $d = p$ in Theorems 8, 14 and 15. We fix $\sigma = 8$ as in the binary case. Let ct_1 and ct_2 denote the two ciphertexts obtained by packing a part of or whole $\mathbf{A}$ and $\mathbf{B}$, respectively. For example, in our second packing method, we have $\mathsf{ct}_1 = \mathsf{ct}_{\mathrm{mat}}^{(1)}(\mathbf{A})$ and $\mathsf{ct}_2 = \mathsf{ct}_{\mathrm{mat}}^{(2)}(\mathbf{B})$. In every method, each coefficient of the decryption polynomial of $\mathsf{ct} = \mathsf{ct}_1 * \mathsf{ct}_2$ is not greater than mp (see [75, Section 3.2] for details). Hence, in every method, it is sufficient to take $t = mp + 1$ as the plaintext modulus. For the degree parameter n, the previous method requires $n \geq 2mp$ by Theorem 8, our first method $n \geq 2mp(m+1)$ by Theorem 14, and our second method $n \geq 2p(m^3 + m + 1)$ by Theorem 15. In our experiments, we take $m = 16$ and $p = 10$, and hence we fix $t = mp + 1 = 161 < 2^8$. In this setting, we take $n = 2^{13} = 8192 \geq 2mp(m+1)$ and $n = 2^{17} = 131072 \geq 2p(m^3 + m + 1)$ in our first and second methods, respectively. When we set $n = 2^{17}$, by inequality (3.7), it requires $q \geq 8n^2t^2\sigma^4 \approx 2^{3+34+16+12} = 2^{65}$ in order to avoid decryption failure of ct. In our experiments, we took one prime q of 70-bit in every method for a margin. For such q, it requires $n \geq 2048$ for more than 80-bit security as in the binary case, and hence we set $n = 2048 \geq 2mp$ in the previous method.

Multiple Matrix Multiplication Consider ℓ matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ with size $M \times M$ and p-bit entries (in our experiments, we take $\ell = 2, 3$ and 4). In our experiments, we divide every $M \times M$ matrix $\mathbf{A}_i$ for $\ell = 2$ (resp. $\ell = 3, 4$) into sub-matrices with size 16×16 (resp. 8×8). From Theorem 17, we set $n = 16^3 = 4096$ (resp., $n = 8^4 = 4096$, $n = 8^5 = 32768$) for the case $\ell = 2$ (resp., $\ell = 3, 4$). Since every

coefficient of $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ is at most $M^{\ell-1}2^{\ell p}$, it needs to take $t \geq M^{\ell-1}2^{\ell p}$ for obtaining the correct decryption result. By using our CRT method with number k, we can divide the parameter t as $t = \prod_{i=1}^{k} t_i$ for small t_i's with $t_1 \approx t_2 \approx \cdots \approx t_k$. Hence it only requires

$$t_i \geq \left(M^{\ell-1}2^{\ell p}\right)^{1/k} \tag{3.8}$$

for all $1 \leq i \leq k$ (but it requires k ciphertexts for every sub-matrix of $\mathbf{A}_i$).

Let ct be a ciphertext corresponding to $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ (see Theorem 17). By Lemma 4, it needs to satisfy the inequality $\|\langle \mathsf{ct}, \mathbf{s} \rangle\|_\infty < q/2$ to avoid decryption failure. Let U be an upper bound of the ∞-norm size of $\langle \mathsf{ct}_0, \mathbf{s} \rangle$ for any *fresh* ciphertext ct_0. By Theorem 17, it *mainly* requires $\ell - 1$ homomorphic multiplications for obtaining ct from fresh (packed) ciphertexts of $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ (actually, it requires additional homomorphic additions due to our block-matrix method). Therefore we roughly estimate $\|\langle \mathsf{ct}, \mathbf{s} \rangle\|_\infty \leq n^{\ell-1}U^\ell$ by the well-known fact that $\|a \cdot b\|_\infty \leq n \cdot \|a\|_\infty \cdot \|b\|_\infty$ for any $a, b \in R_q = \mathbb{Z}_q[x]/(x^n + 1)$ (here we ignore the noise by homomorphic additions since such noise is negligibly smaller than that by homomorphic multiplications). According to the experimental estimation of [75], we can take $U = 2t_i\sigma^2\sqrt{n}$ (recall that we use t_i instead of t by our CRT method). In our experiments, we take the same prime q for all the plaintext parameters t_i. Hence it is sufficient to take a prime q roughly satisfying

$$2n^{\ell-1}U^\ell = 2n^{\ell-1}\left(2t_i\sigma^2\sqrt{n}\right)^\ell < q. \tag{3.9}$$

As in [62, 28], we always fix $\sigma = 8$ in our experiments.

Example 18. *Here we consider $M = 32$ and $p = 16$. Set $k = 1$ (i.e., we do not use our CRT method). The case $\ell = 2$ (resp. $\ell = 3$) requires $t \geq 2^{37}$ and $q > 2^{113}$ (resp. $t \geq 54$ and $q > 226$) from Inequalities (3.8) and (3.9) by setting $n = 4096$. See Table 3.3 below for our chosen parameters.*

Security of Our Selected Parameters

The computational complexity of the LWE assumption can be measured by the *Hermite rootfactor* δ. Given parameters (n, q, t, σ), the Hermite root factor (for the distinguishing attack against LWE) can be calculated as

$$\lg(\delta) = \frac{1}{n \cdot \lg(q)} \left(\frac{1}{2} \lg \left(\frac{c \cdot q}{\sigma} \right) \right)^2, \tag{3.10}$$

as in [55], where $c \approx \sqrt{\lg(1/\epsilon)/\pi}$ with advantage ϵ. We take $c = 3.758$ for $\epsilon = 2^{-64}$ as in [62]. In general, the Hermite root factor $\delta < 1.006$ is secure for the LWE assumption with 80-bit security (with enough margin). In particular, we make use of our CRT method with number k for flexibly choosing parameters with both enough security and practical performance (see the below example).

Example 19. *In this example, we consider the case of* $\ell = 2$, $M = 32$ *and* $p = 32$. *When we do not use our CRT method (i.e., we set* $k = 1$ *as the CRT number), we need to take* $t \geq 2^{69}$ *and* $q > 2^{177}$ *from Inequalities* (3.8) *and* (3.9) *by setting* $n = 4096$. *This parameter setting gives* $\delta > 1.00742$ *from the Equation* (3.10). *In contrast, when we use our CRT method with* $k = 2$, *we can take* $t_1 \approx t_2 \approx 2^{35}$ *from* (3.8), *and then it requires* $q > 2^{113}$ *from* (3.9). *When we take an* 113*-bit prime for* q, *we have* $\delta \approx 1.0045$.

Implementation Results

In this subsection, we provide implementation results of our packing method over the BV scheme for secure matrix multiplications. Our experiments ran on an Intel(R) Xeon(R) CPU E5-46170@2.90GHz, using PARI library [68] (version 2.9.2) in C programs.

Single Matrix Multiplication

In this subsection, we write the implementation results of single matrix multiplication for binary and non-binary entries.

Binary Case Our chosen parameters and implementation results for the binary case are given in Table 3.1. As shown in Table 3.1, our two methods are much faster

Table 3.1 Performance of secure matrix multiplication of $m \times m$ matrices with binary entries

(In seconds)	m	$(n, \log_2(q), t, \sigma)$	Encryption	Secure Matrix Mul.	Decryption	Total time
Previous method [75]	16	$(2048, 60, 17, 8)$	0.3987	11.5704	3.5352	15.5043
	32	$(\mathbf{2048, 60, 33, 8})$	**0.7950**	**47.8745**	**14.063**	**62.7325**
Our first method	16	$(2048, 60, 17, 8)$	0.0150	0.3064	0.2170	0.5384
	32	$(\mathbf{2048, 60, 33, 8})$	**0.0160**	**0.5780**	**0.3592**	**0.9532**
Our second method	16	$(4096, 60, 17, 8)$	0.0439	0.0438	0.0328	0.1205
	32	$(\mathbf{32768, 60, 33, 8})$	**0.1814**	**0.5220**	**0.4036**	**1.1070**

Table 3.2 Performance of secure matrix multiplication of 16×16 matrices with entries less than $p = 10$ bits

(In seconds)	$(n, \log_2(q), t, \sigma)$	Encryption	Secure Matrix Mul.	Decryption	Total time
Previous method [75]	$(2048, 70, 161, 8)$	1.4454	9.4483	3.6581	14.5518
Our first method	$(8192, 70, 161, 8)$	1.5996	1.5797	0.7624	3.9417
Our second method	$(131072, 70, 161, 8)$	4.2024	2.0834	0.9797	7.2655

than the previous method [74]. Specifically, our first method is about $2m$ times faster than the previous method [74]. For $m = 16$, our second method is several times faster than the first one. However, for $m = 32$, our first method is slightly faster than the second one in total. This is due to that our second method requires $n \geq m^3 = 32^3 = 32768$ while our first method requires only $n \geq m^2 = 32^2 = 1024$ (as in [74], we took $n \geq 2048$ for enough security). Then we conclude that our second method is efficient for $m \leq 32$, but our first method is more efficient for $m \geq 64$ in secure multiplications of $m \times m$ matrices with binary entries.

Non-Binary Case Our chosen parameters and implementation results for the non-binary case are given in Table 3.2. As we can see from the Table 3.2 our first method is around 5-times faster and the second method is around 2-times faster than the previous method. In this case, we are not using the same parameter for our both the methods as in the previous method. Our first and second methods require $n = 8192 \geq 2mp(m + 1)$ and $n = 131078 \geq mp(m^3 + m + 1)$ respectively. Because of the very large n in our second method, our first method is faster than the second method. However, in the sense of computation efficiency, our second method is better than the first method because we need m-homomorphic multiplications for our first method and just only one homomorphic multiplication for our second

Table 3.3 Performance (seconds) of secure matrix multiplication between with size $M \times M$ and p-bit entries.. We split t as $t = \prod_{i=1}^{k} t_i$ with $t_1 \approx \cdots \approx t_k$ by our CRT method, and we use a prime q for all t_i's)

Integral entry	m	p	b	CRT	$(\lg(q), \lg(t_i))$	Encryption	Sec Matrix Mul.	Decryption	Total Time
$\ell = 2, n = 4096$	$M = 16$	16	1	1	$(113, 37)$	0.0217	0.0341	0.0237	0.0795
		32	1	2	$(113, 35)$	0.043	0.0706	0.0522	0.1658
	$M = 32$	16	2	1	$(113, 37)$	0.0879	0.31	0.1154	0.5132
		32	2	2	$(113, 35)$	0.1894	0.6203	0.2129	1.0227
$\ell = 3, n = 4096$	$M = 16$	16	2	2	$(148, 28)$	0.2195	1.8465	0.484	2.5501
		32	2	4	$(148, 26)$	0.4357	3.6457	0.998	5.0793
	$M = 32$	16	4	2	$(148, 28)$	0.8612	14.7657	2.3035	17.9303
		32	4	4	$(148, 26)$	1.7122	29.0408	4.6039	35.3569
$\ell = 4, n = 32768$	$M = 16$	16	2	1	$(400, 74)$	2.4098	40.3412	8.4936	51.2446
		32	2	1	$(660, 138)$	3.8877	68.6321	14.4308	86.9506
	$M = 32$	16	4	1	$(400, 76)$	9.4005	329.0052	36.4381	374.8438
		32	4	4	$(660, 140$	15.8673	567.3074	61.9360	645.1106

method. Once we increase the size of entries and dimension of matrices, our first method will be much faster.

Multiple Matrix Multiplications

In this subsection, we write the implementation result for secure matrix multiplications for both cases: integer and real entries. In Table 3.3, we show our chosen parameters of the BV scheme and running time for secure multiplications among two, three and four matrices. For two matrices with 16×16 size and 10-bit entries, our packing method for single matrix multiplication needed to set $n = 131072$ from Theorem 10 of [28], and it took about 7.27 seconds from [28, Table 2]. We divide every $M \times M$ matrix $\mathbf{A}_i$ for $\ell = 2$ (resp. $\ell = 3, 4$) into sub-matrices with size 16×16 (resp. 8×8) by our block-matrix method, and then we set $n = 4096$ (resp. $4096, 32768$) for $\ell = 2$ (resp. $\ell = 3, 4$) from Theorem 3.2.2. From Table 3.3, our method took only about 0.0795 (resp., 0.1658) seconds for $M = 16$ and p-bit entries for $p = 16$ (resp., $p = 32$) and $\ell = 2$. Our method with modifications is much faster than [28] due to taking smaller n, and our method for large p does not reduce the performance significantly due to our CRT method. Actually, our CRT method enables us to take small size of plaintext space and then we can take practical q for large p (however, our CRT method requires more ciphertexts and additional homomorphic operations).

3.3.2 Performance over BGV Scheme

In this section, we show implementation results for our secure matrix multiplications (described in Subsection 3.2.2) over `HElib` [46]. 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 $p = 16, 32$.

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 [44, 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 3.4 below for our chosen parameters.)

Table 3.4 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

Experiment and Performance Results

Experiment Our experiments ran on an Intel(R)Xeon(R) CPU E5-46170@2.90GHz, using `HElib` [46] 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 (2.3) for efficiency (see [37, Section 3]).

Performance Results In Table 3.4, 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 $\ell = 2, 3, 4$:

1. For the case $\ell = 2$, our method is the same as Duong et al.'s method [28]. 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 [28, Subsection 4.2] needs $n = 131072$ by [28, Theorem 10], and it took about 7.27 seconds from [28, Table 2]. In contrast, Table 3.4 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 [60]. From [60, 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 [68] (version 2.9.2) in `C` programs for implementing the BV scheme [12]. In this full version paper, we use $M = 8$ as the size of sub-matrices (cf., $M = 16$ is used in [60]), and hence smaller n is sufficient. From Table 3.4, 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 [60, 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 3.4, this is about 3 or 4 times slower than the case $\ell = 3$. However, for larger ℓ, the performance might be considerably slower since a huge n is required in our method. Big jumping exponents are used in our packed polynomials for large ℓ (see Subsection 3.2.2). See also Subsection 3.3.2 below for advantages and limitations of our method.

Comparison in Matrix-Vector Multiplications

As described in Subsection 3.1.5, 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 3.2.1 and 3.2.2):

Theorem 20. *Let $\ell \geq 4$ and assume $n \geq m^\ell$. Let*

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

where $\mathsf{ct}^{(\ell)}(\mathbf{v}) = Enc\left(pm_\ell^{(2)}(\mathbf{v}), \mathsf{pk}\right)$ is a packed ciphertext of $\mathbf{v}$ (see also Subsection 3.2.2 for other packed ciphertexts). Let $Dec(\mathsf{ct}, \mathsf{sk}) \in R_t$ denote its decryption polynomial. If decryption is correct for 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 $Dec(\mathsf{ct}, \mathsf{sk})$.

Performance Comparison

In Table 3.5, 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 3.5. We also used the hoisting optimization mentioned in [47] to speed further 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 3.3.2. For secure matrix-vector multiplications, we set larger M as the size of every sub-matrix than Table 3.4, 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 efficient and true rotations over slots (see [45, 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 21. *For $\ell = 4$, $m = 32$ and $p = 32$ in Table 3.5, 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 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 much 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 3.2, 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 3.5. In the following, we shall discuss advantages and limitations of our method (the same discussion holds for secure matrix multiplications):

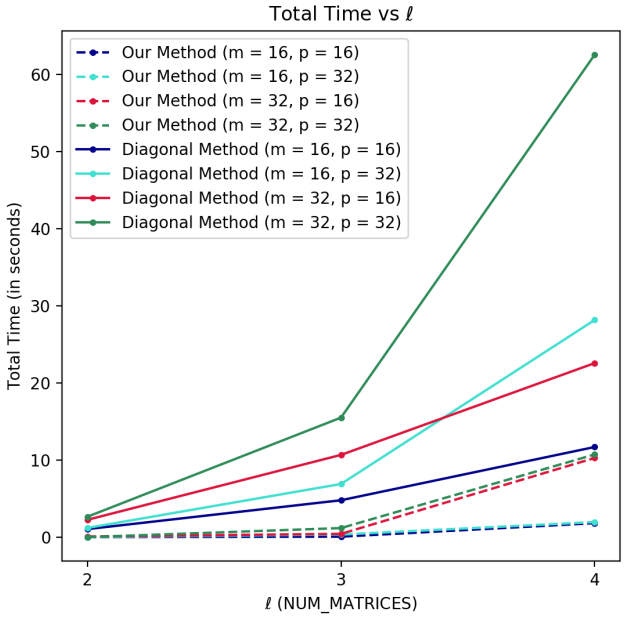

Fig. 3.2 Plot showing comparison of the results between our method and the diagonal-based method for secure matrix-vector multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_{\ell-1} \times \mathbf{v}$. Dashed lines and solid lines are used to represent the total time of our method and the diagonal-based method, respectively (the data are given from Table 3.5).

Advantages As seen from Table 3.5 and Figure 3.2, 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 3.2, 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 3.2 for the diagonal-based method is much larger than ours for larger ℓ. We also see from Table 3.5 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. 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 [45]).

Table 3.5 Performance comparison of our method (Theorem 20) with the diagonal-based method (Subsection 3.1.5) 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**

Chapter 4

Application for Predictive Analysis

The significant advancements in the field of homomorphic encryption have led to a grown interest in securely outsourcing data and computation for critical privacy applications. In this chapter, we focus on the problem of performing secure predictive analysis, such as principal component analysis (PCA) and linear regression, through *exact* arithmetic over encrypted data. We improve the plaintext structure of Lu et al.'s protocols (from NDSS 2017), by switching over from *linear* array arrangement to a *two*-dimensional hypercube. This enables us to utilise the SIMD (Single Instruction Multiple Data) operations to a larger extent, which results in improving the space and time complexity by a factor of matrix dimension. We implement both Lu et al.'s method and ours for PCA and linear regression over `HElib`, a software library that implements the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme. In particular, we show how to choose the optimal parameters of the BGV scheme for both methods. For example, our experiments show that our method takes 45 seconds to train a linear regression model over a dataset with 32k records and 6 numerical attributes, while Lu et al.'s method takes 206 seconds.

4.1 Introduction

In the recent years, the cloud computing paradigm has grown in popularity as an economical solution for outsourcing data and computation. It enables ubiquitous access to shared storage and computational resources over the internet, and hence it is being

adopted by many organisations. However, storing data on the cloud raises security and privacy concerns, since the cloud service provider can not only access the data but also share it with other parties. This makes it difficult to keep control of the data for applications that have privacy as a principal concern. A great solution to address these concerns is homomorphic encryption that enables computation on encrypted data.

In this work, we use leveled-FHE for performing predictive statistics such as PCA and linear regression, through *exact* arithmetic over encrypted data (cf., approximate arithmetic). We choose a variant [37] of the leveled BGV scheme [10] as our cryptosystem, and leverage the software library `HElib` [44] for its implementation. Many works have been proposed to address the problem of performing statistical analysis over encrypted data (see [42, 71, 8, 57]). The solution of [8] only addresses model evaluation, while the methods of [42, 71] can only perform statistics on data with very low dimension. Recently, Lu et al. [57] proposed a solution to perform PCA and linear regression on data with up to 20 numerical attributes. Their solution achieves much better results than any previous work by utilising the linear array structure of the plaintext slots. Our main aim is to improve Lu et al.'s method for further efficiency. Our contribution in this work is two-fold:

1. Firstly, we improve upon the plaintext structure used in Lu et al.'s method [57] by utilising a *two*-dimensional hypercube structure. This gives us benefits in terms of both space and time complexity. As a result, we reduce the complexity of their methods by a factor of data dimension. In the process, we also develop some general-purpose procedures for matrix operations that are significantly faster than the previously known solutions in `HElib`.

2. Secondly, we address the problem of optimal parameter selection in `HElib`, which is non-trivial for our application and was not discussed in [57] carefully. In this work, we describe how to choose optimal parameters for our method as well as Lu et al.'s. We also compare the performance of our method with Lu et al.'s method for performing PCA and linear regression over `HElib`.

4.1.1 Principal Component Analysis (PCA)

PCA is a dimensionality-reduction tool, that takes a number of possibly correlated variables and transforms them into a smaller number of uncorrelated variables, while retaining most of the information. These uncorrelated variables are called principal components, and they do not necessarily have a physical interpretation. PCA can be seen as a rotation of the original axes to a new set of orthogonal axes, that are aligned in the direction of maximum variation.

Let $\mathbf{X}$ be a data matrix, with N records and d numerical attributes, which is defined as:

$$\mathbf{X} = \underbrace{\left.\begin{pmatrix} — & \mathbf{x}_0^\mathsf{T} & — \\ & \vdots & \\ — & \mathbf{x}_{N-1}^\mathsf{T} & — \end{pmatrix}\right\}}_{d \text{ attributes}} N \text{ records}$$

The problem of finding the principal components of $\mathbf{X}$ is the same as finding the eigenvectors of its covariance matrix $\mathbf{\Sigma}$, which is defined as:

$$\mathbf{\Sigma} = \frac{1}{N}\mathbf{X}^\mathsf{T}\mathbf{X} - \boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T}, \text{ where } \boldsymbol{\mu}^\mathsf{T} = \frac{1}{N}\sum_{i=0}^{N-1}\mathbf{x}_i^\mathsf{T}.$$

To find the eigen vectors of $\mathbf{\Sigma}$, a technique called the Power Method is used. PowerMethod is an iterative technique that is used to find the dominant eigen-vector of a matrix, and is described in Algorithm 1. The intuition behind the Power Method is that by multiplying

Algorithm 1 PowerMethod$(\mathbf{\Sigma}, T)$

Input:

- $\mathbf{\Sigma}$: covariance matrix of $\mathbf{X}$
- T: number of iterations

Output:

- $\mathbf{u}_1, \lambda_1$: dominant eigen-vector of $\mathbf{\Sigma}$ and its eigen-value

1: Choose a random vector $\mathbf{v}^{(0)}$ of size d
2: **for** $i = 1$ to T **do**
3: $\quad \mathbf{v}^{(i)} = \mathbf{\Sigma}\mathbf{v}^{(i-1)}$
4: **end for**
5: return $\mathbf{u}_1 = \mathbf{v}^{(T)}/\|\mathbf{v}^{(T)}\|$ and $\lambda_1 = \|\mathbf{v}^{(T)}\|/\|\mathbf{v}^{(T-1)}\|$

the initial vector $\mathbf{v}$ repeatedly by $\mathbf{\Sigma}$, $\mathbf{v}$ is stretched in the direction of the $\mathbf{\Sigma}$'s dominant

eigen-vector $\mathbf{u}_1$, to the point where the vector lies almost entirely in the direction of $\mathbf{u}_1$. Power Method can also be used to find the k-th dominant eigen-vector of $\mathbf{\Sigma}$ by using the EigenShift procedure, which is described in Algorithm 2. For input k, the Eigen Shift

Algorithm 2 EigenShift$(\mathbf{\Sigma}, \{\mathbf{u}_i, \lambda_i\}_{i=1}^{k-1})$

Input:

- $\mathbf{\Sigma}$: covariance matrix of $\mathbf{X}$
- $\{\mathbf{u}_i, \lambda_i\}$: i-th dominant eigen-vector of $\mathbf{\Sigma}$ and its eigen-value

Output:

- $\mathbf{\Sigma}_k$: k-shifted covariance matrix of $\mathbf{X}$

1: $\mathbf{\Sigma}_1 = \mathbf{\Sigma}$
2: **for** $i = 1$ to $k - 1$ **do**
3: $\quad \mathbf{\Sigma}_{i+1} = \mathbf{\Sigma}_i - \lambda_i \mathbf{u}_i \mathbf{u}_i^\mathsf{T}$
4: **end for**
5: return $\mathbf{\Sigma}_k$

procedure outputs a matrix $\mathbf{\Sigma}_k$ whose most dominant eigen-vector is $\mathbf{u}_k$, where $\mathbf{u}_k$ is the k-th most dominant eigen-vector of $\mathbf{\Sigma}$ and λ_k is its associated eigen-value. Therefore, combining these two methods, the K dominant eigen-vectors (and their associated eigen-values) of the covariance matrix $\mathbf{\Sigma}$ can be found to get the K principal components of $\mathbf{X}$.

4.1.2 Linear Regression (LR)

Linear Regression is a statistical procedure that models a relationship between the target (dependent) variable y and one or more input (independent) variables $\mathbf{x}$ using a linear equation. Given a dataset $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_i^\mathsf{T}, y_i)\}_{i=0}^{N-1} \in \mathbb{Z}^{N \times d}$, where $\mathbf{x}_i^\mathsf{T}$ are the input variables and y_i is the target variable, the aim is to find a vector of weights $\mathbf{w}$ such that $\mathbf{y} \approx \mathbf{X}\mathbf{w}$. We obtain $\mathbf{w}$ by minimizing the least-squares error defined by:

$$\mathbf{w} = \arg\min_{\mathbf{w}^*} \frac{1}{N} \sum_{i=1}^{N} \|y_i - \mathbf{x}_i^\mathsf{T} \mathbf{w}^*\|^2.$$

The most popular solution is an iterative technique called Gradient Descent. Gradient Descent is useful when we are operating on data with very large dimension d. For smaller dimensions, there is a one-step analytical solution called Normal Equation method, which computes $\mathbf{w}$ as:

$$\mathbf{w} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}.$$

We leverage an iterative division-free technique from [43] for matrix inversion, which only involves matrix multiplications and additions. This technique, described in Algorithm 3, allows us to compute inverse of encrypted matrices. There can be different choices for

Algorithm 3 InvertMatrix$(\mathbf{M}, \alpha, T)$

Input:

- $\mathbf{M}$: given matrix
- α: a constant close to the dominant eigen-value of $\mathbf{M}$
- T: number of iterations

Output:

- $\mathbf{M}^{-1}$: inverse of matrix $\mathbf{M}$

1: $\mathbf{A}^{(0)} = \mathbf{M}, \mathbf{R}^{(0)} = \mathbf{I}$ ▷ $\mathbf{I}$ is the identity matrix
2: $\alpha^{(0)} = \alpha$
3: **for** $i = 1$ to T **do**
4: $\quad \mathbf{R}^{(i)} = 2\alpha^{(i-1)}\mathbf{R}^{(i-1)} - \mathbf{R}^{(i-1)}\mathbf{A}^{(i-1)}$
5: $\quad \mathbf{A}^{(i)} = 2\alpha^{(i-1)}\mathbf{A}^{(i-1)} - \mathbf{A}^{(i-1)}\mathbf{A}^{(i-1)}$
6: $\quad \alpha^{(i)} = \alpha^{(i-1)}\alpha^{(i-1)}$
7: **end for**
8: return $\mathbf{M}^{-1} = \mathbf{R}^{(T)}/\alpha^{(T)}$

the input α to the function InvertMatrix, but taking α close to the largest eigen value of matrix $\mathbf{M}$ ensures a stable quadratic convergence to $\mathbf{M}^{-1}$.

4.2 Matrix Operations

In the previous section, we described statistical procedures that we will compute securely in the cloud. It is evident that these procedures require matrix operations. In this section, we describe the method used in [57] and propose a new method for matrix operations over encrypted matrices. We also draw a complexity comparison between the two methods in terms of homomorphic operations.

Data Movement

The basic data movement operation is rotation and all other operations that move data depend on it (See [45, Section 4] for more details). Rotation comes in two flavours depending on the arrangement of plaintext slots. There are two ways to arrange the plaintext slots:

- *Hypercube* arrangement: As described before, the plaintext slots natively assume the structure of an n-dimensional *hypercube* $\mathbf{V}$ and each slot is indexed by some $\mathbf{e} = (e_1, \ldots, e_n)$, where e_i ranges over $[m_i]$. The dimensions of this *hypercube* can be rotated independently through the rotate1D procedure. A call to "$\texttt{rotate1D}(\mathbf{V}, i, k)$" will move the content of slot $(e_1, \ldots, e_i, \ldots, e_n)$ to the slot $(e_1, \ldots, e_i + k, \ldots, e_n)$ of $\mathbf{V}$, where addition is done modulo m_i.

- *Linear Array* arrangement: In this arrangement, the plaintext slots are presented to the application in the form of a *linear array*. The number of elements in the array is $\ell = |\mathbb{Z}_m^*/\langle t \rangle|$. It is made possible by ordering the slots lexicographically over the vector of indices of *hypercube*. The position k of this *linear array* is identified by $\mathbf{e}^{(k)} = (e_1^{(k)}, \ldots, e_n^{(k)})$, where $k \in [\ell]$. A call to "$\texttt{rotate}(\mathbf{v}, k)$" will move the content of slot j (resp., $\mathbf{e}^{(j)}$) to the slot $j + k$ (resp., $\mathbf{e}^{(j+k)}$) of $\mathbf{v}$, where addition is done modulo ℓ, thereby rotating the array by k positions. The addition over the respective index vectors can be seen as addition with carry over n-digit numbers, where i-th digit has base m_i and n-th digit is least significant.

We have other high level data movement operations such as total-sum and replicate. As mentioned before, these operations depend on rotation. Hence, they have different impact depending on the type of slot arrangement. They are defined as follows:

- For *hypercube* arrangement: Given an n-dimensional *hypercube* $\mathbf{V} = (\mathbf{V}[e_1, \ldots, e_n])$, where e_j ranges over $[m_j]$, the function "$\mathsf{TS1D}(\mathbf{V}, j)$" outputs:

$$\mathbf{W}_{\mathsf{TS}}[e_1, \ldots, e_j, \ldots, e_n] = \textstyle\sum_{k=0}^{m_j - 1} \mathbf{V}[e_1, \ldots, k, \ldots, e_n].$$

  Let $\mathbf{e}^{(i)}$ be a vector of indices excluding the index for j-th dimension i.e. $\mathbf{e}^{(i)} = (e_1^{(i)}, \ldots, e_{j-1}^{(i)}, e_{j+1}^{(i)}, \ldots, e_n^{(i)})$, where i ranges over $[\ell/m_j]$. Given a set $\{k_i\}_{i=0}^{\ell_j - 1}$ with $\ell_j = \ell/m_j$ indices, the function "$\mathsf{replicate1D}\left(\mathbf{V}, j, \{k_i\}_{i=0}^{\ell_j - 1}\right)$" outputs:

$$\mathbf{W}_{\mathsf{replicate}}[e_1^{(i)}, \ldots, e_j, \ldots, e_n^{(i)}] = \mathbf{V}[e_1^{(i)}, \ldots, k_i, \ldots, e_n^{(i)}],$$

where e_j ranges over $[m_j]$. Both "TS1D" and "replicate1D" have a running time of $O(\log m_j)$ rotations and additions.

- For *linear array* arrangement: Given a vector $\mathbf{v} = (\mathbf{v}[i])_{i=0}^{\ell-1}$, the functions "$\mathsf{TS}(\mathbf{v})$" and "$\mathsf{replicate}(\mathbf{v}, k)$" respectively output:

$$\mathbf{w}_{\mathsf{TS}}[j] = \sum_{k=0}^{\ell-1} \mathbf{v}[k] \text{ and } \mathbf{w}_{\mathsf{replicate}}[j] = \mathbf{v}[k].$$

  These functions have a running time of $O(\log \ell)$ rotations and additions.

For details on underlying algorithms, see [45, Section 4].

4.2.1 Lu et al.'s Method (Linear Array Arrangement)

Lu et al. proposed a matrix multiplication method in [57] that assumes a linear array arrangement of plaintext slots, and works along the lines of column-order matrix-vector multiplication method described in [45, Section 4.3]. Given the one-dimensional array arrangement, a vector can be packed directly in the plaintext slots (followed by zeros if the length of vector is less than ℓ) and encrypted. Their proposed techniques work on matrices encrypted in row-order i.e. each row vector is encrypted separately. The encryption of a vector $\mathbf{u} \in \mathbb{Z}_t^d$ and a matrix $\mathbf{X} \in \mathbb{Z}_t^{d \times d}$ are represented as $\mathsf{ct}(\mathbf{u})$ and $\mathbf{C}(\mathbf{X}) = \{\mathsf{ct}(\mathbf{x}_i^\mathsf{T})\}_{i=0}^{d-1}$ respectively.

Outer Product of Vectors

Given ciphertexts of vectors $\mathbf{u}$ and $\mathbf{v}$, to compute their outer product homomorphically, the function described in Algorithm 4 is used.

Matrix-Vector Multiplication

Lu et al. have used the column-order matrix vector multiplication method of [45] for matrix-vector multiplication. They took advantage of the fact that their application only requires dealing with *symmetric* matrices. Therefore, they could use this method, described in Algorithm 5, even on matrices encrypted in row-order.

Algorithm 4 $\mathsf{OuterProduct}(\mathsf{ct}(\mathbf{u}), \mathsf{ct}(\mathbf{v}))$

Input:

- $\mathsf{ct}(\mathbf{u}), \mathsf{ct}(\mathbf{v})$: ciphertexts of vectors $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_t^d$

Output:

- $\mathbf{C}(\mathbf{uv})$: ciphertexts for rows of matrix $\mathbf{uv} \in \mathbb{Z}_t^{d \times d}$

1: **for** $i = 0$ to $d - 1$ **do**
2: $\quad \mathsf{ct}(\mathbf{uv}_i^{\mathsf{T}}) = \mathsf{replicate}(\mathsf{ct}(\mathbf{u}), i) \cdot \mathsf{ct}(\mathbf{v})$
3: **end for**
4: return $\mathbf{C}(\mathbf{uv})$ ▷ $\mathbf{C}(\mathbf{uv}) = \{\mathsf{ct}(\mathbf{uv}_i)\}_{i=0}^{d-1}$

Algorithm 5 $\mathsf{MatVecMul}(\mathbf{C}(\mathbf{X}), \mathsf{ct}(\mathbf{u}))$

Input:

- $\mathbf{C}(\mathbf{X})$: ciphertexts for rows of a *symmetric* matrix $\mathbf{X} \in \mathbb{Z}_t^{d \times d}$
- $\mathsf{ct}(\mathbf{u})$: ciphertext of vector $\mathbf{u} \in \mathbb{Z}_t^d$

Output:

- $\mathsf{ct}(\mathbf{Xu})$: ciphertext of vector $\mathbf{Xu} \in \mathbb{Z}_t^d$

1: **for** $i = 0$ to $d - 1$ **do**
2: $\quad \mathsf{ct}(\mathbf{Xu}) \mathrel{+}= \mathsf{ct}(\mathbf{x}_i^{\mathsf{T}}) \cdot \mathsf{replicate}(\mathsf{ct}(\mathbf{u}), i)$
3: **end for**
4: return $\mathsf{ct}(\mathbf{Xu})$

Matrix Addition & Multiplication

Matrix addition and multiplication are computed in such a way that the layout is preserved i.e. output matrix is also encrypted in row order. Matrix addition is fairly straight-forward, and is done by adding the corresponding rows of the two matrices. Matrix multiplication is performed by the MatMul function described in Algorithm 6.

4.2.2 Our Method (Hypercube Arrangement)

We leverage a *two*-dimensional ($n = 2$, ref. Section 2.3.3) *hypercube* arrangement of plaintext slots to optimize matrix operations over encrypted matrices and vectors. Let the size of the *first* and the *second* dimension be $m_1 = \mathsf{ord}(f_1)$ and $m_2 = \mathsf{ord}(f_2)$ respectively. The *hypercube* can be seen as a matrix having m_1 rows and m_2 columns. For the rest of discussion, we treat our *hypercube* arrangement of plaintext slots as a $m_1 \times m_2$ matrix $\mathbf{V}$.

Algorithm 6 $\mathsf{MatMul}(\mathbf{C}(\mathbf{X}), \mathbf{C}(\mathbf{Y}))$

Input:

– $\mathbf{C}(\mathbf{X}), \mathbf{C}(\mathbf{Y})$: ciphertexts for rows of matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{Z}_t^{d \times d}$

Output:

– $\mathbf{C}(\mathbf{XY})$: ciphertexts for rows of matrix $\mathbf{XY} \in \mathbb{Z}_t^{d \times d}$

1: **for** $i = 0$ to $d - 1$ **do**
2: **for** $j = 0$ to $d - 1$ **do**
3: $\mathsf{ct}(\mathbf{xy}_i^\mathsf{T})$ += $\mathsf{replicate}(\mathsf{ct}(\mathbf{x}_i^\mathsf{T}), j) \cdot \mathsf{ct}(\mathbf{y}_j^\mathsf{T}))$
4: **end for**
5: **end for**
6: return $\mathbf{C}(\mathbf{XY})$ ▷ $\mathbf{C}(\mathbf{XY}) = \{\mathsf{ct}(\mathbf{xy}_i^\mathsf{T})\}_{i=0}^{d-1}$

A matrix $\mathbf{X} \in \mathbb{Z}_t^{d \times d}$ can be packed in $\mathbf{V}$ by putting the (i, j)-th entry of the matrix in (i, j)-th position of the hypercube i.e. $\mathbf{V}[i, j] = x_{i,j}$ (all other positions are filled with zeros). There are two ways a vector $\mathbf{u} \in \mathbb{Z}_t^d$ can be packed in $\mathbf{V}$:

1. *Row-wise* packing: If $\mathbf{u}$ is supposed to act like a row-vector in the computation, then we pack the vector $\mathbf{u}$ (row-wise) in all the rows of $\mathbf{V}$ i.e. $\mathbf{V}[i, j] = u_j$, where $i \in [m_1]$ and $j \in [d]$. Given $\mathbf{u} = \{u_j\}_{j=0}^{d-1}$, we have:

$$\mathbf{V} = \left.\begin{bmatrix} u_0 & \dots & u_{d-1} \\ \vdots & \ddots & \vdots \\ u_0 & \dots & u_{d-1} \end{bmatrix}\right\}_{m_1}$$

2. *Column-wise* packing: Similarly, if $\mathbf{u}$ is supposed to act like a column-vector in the computation, then we pack the vector $\mathbf{u}$ (column-wise) in all the columns of $\mathbf{V}$ i.e. $\mathbf{V}[i, j] = u_i$, where $i \in [d]$ and $j \in [m_2]$. Given $\mathbf{u} = \{u_i\}_{i=0}^{d-1}$, we have:

$$\mathbf{V} = \underbrace{\begin{bmatrix} u_0 & \dots & u_0 \\ \vdots & \ddots & \vdots \\ u_{d-1} & \dots & u_{d-1} \end{bmatrix}}_{m_2}$$

There are two major advantages of using the *hypercube* arrangement:

- *Space*-efficient: Using the *hypercube* arrangement, we can pack the whole matrix in a single plaintext, as opposed to the *linear array* arrangement, which requires d plaintexts. Let $\mathbf{V}_l = \{\mathbf{v}_i^\mathsf{T}\}_{i=0}^{d-1}$ and $\mathbf{V}_h$ be the packing of matrix $\mathbf{X} \in \mathbb{Z}_t^{d \times d}$ in the *linear array* and *hypercube* arrangement respectively. Then, we have:

$$\underbrace{\begin{bmatrix} \begin{bmatrix} x_{0,0} & \dots & x_{0,d-1} \end{bmatrix} \\ \vdots \quad \ddots \quad \vdots \\ \begin{bmatrix} x_{d-1,0} \dots x_{d-1,d-1} \end{bmatrix} \end{bmatrix}}_{\mathbf{V}_l = \{\mathbf{v}_i^\mathsf{T}\}_{i=0}^{d-1}} \Bigg| \underbrace{\begin{bmatrix} x_{0,0} & \dots & x_{0,d-1} \\ \vdots & \ddots & \vdots \\ x_{d-1,0} \dots x_{d-1,d-1} \end{bmatrix}}_{\mathbf{V}_h}$$

d-ciphertexts | 1-ciphertext

- *Time*-efficient: The *hypercube* arrangement offers a huge advantage by enabling us to operate on a dimension, independent of other dimensions. Therefore, given a set of positions $\{k_i\}_{i=0}^{d-1}$, we can replicate the k_i-th element of the i-th row through one replication operation with *hypercube* arrangement as opposed to d replication operations in case of *linear array*. Hence, we can utilize the SIMD operations to a greater extent. On performing replication, we get:

$$\underbrace{\begin{bmatrix} \begin{bmatrix} x_{0,k_0} & \dots & x_{0,k_0} \end{bmatrix} \\ \vdots \quad \ddots \quad \vdots \\ \begin{bmatrix} x_{d-1,k_{d-1}} \dots x_{d-1,k_{d-1}} \end{bmatrix} \end{bmatrix}}_{\left\{\mathsf{replicate}(\mathbf{v}_i^\mathsf{T}, k_i)\right\}_{i=0}^{d-1}} \Bigg| \underbrace{\begin{bmatrix} x_{0,k_0} & \dots & x_{0,k_0} \\ \vdots & \ddots & \vdots \\ x_{d-1,k_{d-1}} \dots x_{d-1,k_{d-1}} \end{bmatrix}}_{\mathsf{replicate1D}\left(\mathbf{V}_h, 2, \{k_i\}_{i=0}^{d-1}\right)}$$

d-replications | 1-replication

This property holds for all data movement operations such as rotation and total-sum. Similarly, we can also replicate the k_i-th element of the i-th column through a call to the function $\mathsf{replicate1D}\Big(\mathbf{V}_h, 1, \{k_i\}_{i=0}^{d-1}\Big)$.

We use the notation $\mathsf{ct}_1(\mathbf{u})$ and $\mathsf{ct}_2(\mathbf{u})$ to denote ciphertexts encrypting the vector $\mathbf{u} \in \mathbb{Z}_t^d$ row-wise and column-wise respectively. The encryption of matrix $\mathbf{X} \in \mathbb{Z}_t^{d \times d}$ is denoted by the notation $\mathsf{ct}(\mathbf{X})$.

Outer Product of Vectors

The outer product of vectors $\mathbf{u} \in \mathbb{Z}_t^d$ and $\mathbf{v} \in \mathbb{Z}_t^d$ is computed by the function defined in Algorithm 7.

Algorithm 7 $\mathsf{OuterProduct1D}(\mathsf{ct}_2(\mathbf{u}), \mathsf{ct}_1(\mathbf{v}))$

Input:

- $\mathsf{ct}_2(\mathbf{u})$: column-wise encryption of $\mathbf{u} \in \mathbb{Z}_t^d$
- $\mathsf{ct}_1(\mathbf{v})$: row-wise encryption of $\mathbf{v} \in \mathbb{Z}_t^d$

Output:

- $\mathsf{ct}(\mathbf{uv})$: ciphertext of matrix $\mathbf{uv} \in \mathbb{Z}_t^{d \times d}$

1: $\mathsf{ct}(\mathbf{uv}) = \mathsf{ct}_2(\mathbf{u}) \cdot \mathsf{ct}_1(\mathbf{v})$
2: return $\mathsf{ct}(\mathbf{uv})$

Matrix-Vector Multiplication

To multiply a matrix $\mathbf{X} \in \mathbb{Z}_t^{d \times d}$ with a vector $\mathbf{u} \in \mathbb{Z}_t^d$, our technique requires the matrix and the vector to be encrypted in one of the following ways:

1. $\mathbf{X}$ is encrypted and $\mathbf{u}$ is encrypted row-wise.
2. $\mathbf{X}^\mathsf{T}$ is encrypted and $\mathbf{u}$ is encrypted column-wise.

It turns out that our technique outputs vector $\mathbf{Xu}$ encrypted column-wise if the input vector $\mathbf{u}$ is encrypted row-wise and vice-versa. This does not pose a problem since we can easily convert between the two packings by multiplying the vector with an identity matrix $\mathbf{I}$. For our application, we can do successive multiplications (without a need for conversion) with encryption of $\mathbf{X}$ since we only deal with *symmetric* matrices. Our

Algorithm 8 $\mathsf{MatVecMul1D}(\mathsf{ct}(\mathbf{X}), \mathsf{ct}_{1/2}(\mathbf{u}))$

Input:

- $\mathsf{ct}(\mathbf{X})$: ciphertext of a *symmetric* matrix $\mathbf{X} \in \mathbb{Z}_t^{d \times d}$
- $\mathsf{ct}_{1/2}(\mathbf{u})$: row/column-wise encryption of $\mathbf{u} \in \mathbb{Z}_t^d$

Output:

- $\mathsf{ct}_{2/1}(\mathbf{Xu})$: column/row-wise encryption of $\mathbf{Xu} \in \mathbb{Z}_t^d$

1: $\mathsf{ct}_{1/2}(\mathbf{Xu}^*) = \mathsf{ct}(\mathbf{X}) \cdot \mathsf{ct}_{1/2}(\mathbf{u})$
2: $\mathsf{ct}_{2/1}(\mathbf{Xu}) = \mathsf{TS1D}(\mathsf{ct}_{1/2}(\mathbf{Xu}^*), 2)$
3: return $\mathsf{ct}_{2/1}(\mathbf{Xu})$ ▷ $\mathsf{ct}_{1/2}(\cdot) \rightarrow \mathsf{ct}_{2/1}(\cdot)$

technique for matrix vector multiplication is defined in Algorithm 8 and the following examples demonstrate its working:

$$\underbrace{\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}}_{\mathsf{ct}(\mathbf{X})} \cdot \underbrace{\begin{bmatrix} e & g \\ e & g \end{bmatrix}}_{\mathsf{ct}_1(\mathbf{u})} \rightarrow \underbrace{\begin{bmatrix} e & 2g \\ 2e & 4g \end{bmatrix}}_{\mathsf{ct}_1(\mathbf{Xu}^*)} \xrightarrow{\mathsf{TS1D}(2)} \underbrace{\begin{bmatrix} e+2g & e+2g \\ 2e+4g & 2e+4g \end{bmatrix}}_{\mathsf{ct}_2(\mathbf{Xu})}$$

$$\underbrace{\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}}_{\mathsf{ct}(\mathbf{X})} \cdot \underbrace{\begin{bmatrix} e & e \\ g & g \end{bmatrix}}_{\mathsf{ct}_2(\mathbf{u})} \rightarrow \underbrace{\begin{bmatrix} e & 2e \\ 2g & 4g \end{bmatrix}}_{\mathsf{ct}_2(\mathbf{Xu}^*)} \xrightarrow{\mathsf{TS1D}(1)} \underbrace{\begin{bmatrix} e+2g & 2e+4g \\ e+2g & 2e+4g \end{bmatrix}}_{\mathsf{ct}_1(\mathbf{Xu})}$$

Note: We can switch between the two encrypted vector packings at the cost of a replication by using the $\mathsf{SwitchOrder1D}$ routine that comes directly from the $\mathsf{MatVecMul1D}$ function by replacing $\mathsf{ct}(\mathbf{X})$ with $\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Therefore, combining the two functions, we get a general-purpose function to do matrix vector multiplication.

Matrix Addition & Multiplication

Matrix Addition is trivial for our packing and can be simply done by adding the ciphertexts of the two matrices. For matrix multiplication, we use the Fox Matrix multiplication method for hypercubes described in [32]. This method can be seen as an extension of the diagonal order matrix-vector multiplication described in [45, Section 4.3]. Suppose we want to multiply a matrix $\mathbf{X} \in \mathbb{Z}_t^{d \times d}$ with a matrix $\mathbf{Y} \in \mathbb{Z}_t^{d \times d}$. Given the hypercube

structure, we can rotate the column vectors of $\mathbf{Y}$ by i positions, denoted by $\mathbf{Y} \lll_1 i$, with one rotation operation. Using one replication operation, we can extract the i-th diagonal of $\mathbf{X}$, defined as $\mathbf{d}_i(\mathbf{X}) = (x_{0,i}, x_{1,i+1}, \ldots, x_{d-1,i-1})$, and replicate it along the rows. Then, we can compute the product $\mathbf{XY}$ as $\sum_{i=0}^{d-1} \mathbf{d}_i(\mathbf{X}) \times (\mathbf{Y} \lll_1 i)$. The MatMul1D function implements our matrix multiplication procedure and is defined in Algorithm 9.

Algorithm 9 $\mathsf{MatMul1D}(\mathsf{ct}(\mathbf{X}), \mathsf{ct}(\mathbf{Y}))$

Input:

– $\mathsf{ct}(\mathbf{X}), \mathsf{ct}(\mathbf{Y})$: ciphertexts of matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{Z}_t^{d \times d}$

Output:

– $\mathsf{ct}(\mathbf{XY})$: ciphertext of matrix $\mathbf{XY} \in \mathbb{Z}_t^{d \times d}$

1: **for** $i = 0$ to $d - 1$ **do**
2: **for** $j = 0$ to $d - 1$ **do**
3: $k_j = i + j \pmod d$
4: **end for**
5: $\mathsf{ct}_2(\mathbf{d}_i(\mathbf{X})) = \mathsf{replicate1D}(\mathsf{ct}(\mathbf{X}), 2, \{k_j\}_{j=0}^{d-1})$
6: $\mathsf{ct}(\mathbf{Y} \lll_1 i) = \mathsf{rotate1D}(\mathsf{ct}(\mathbf{Y}), 1, -i)$
7: $\mathsf{ct}(\mathbf{XY}) \mathrel{+}= \mathsf{ct}_2(\mathbf{d}_i(\mathbf{X})) \cdot \mathsf{ct}(\mathbf{Y} \lll_1 i)$
8: **end for**
9: return $\mathsf{ct}(\mathbf{XY})$

The following example demonstrates the working of MatMul1D:

$$\underbrace{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}_{\mathsf{ct}(\mathbf{X})} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{\mathsf{ct}(\mathbf{Y})} \rightarrow \overbrace{\underbrace{\begin{bmatrix} 1 & 1 \\ 4 & 4 \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{i=0}}^{\mathsf{ct}(\mathbf{d}_0(\mathbf{X}) \times (\mathbf{Y} \lll_1 0))} + \overbrace{\underbrace{\begin{bmatrix} 2 & 2 \\ 3 & 3 \end{bmatrix} \cdot \begin{bmatrix} g & h \\ e & f \end{bmatrix}}_{i=1}}^{\mathsf{ct}(\mathbf{d}_1(\mathbf{X}) \times (\mathbf{Y} \lll_1 1))}$$

Remark 22. *Our matrix multiplication technique defined for square matrices can be easily extended to general matrices. Every matrix can be partitioned into square sub-matrices called blocks. We can then multiply the matrices block-wise using the Block Matrix Multiplication technique.*

Table 4.1 Complexity of Matrix Operations in terms of Homomorphic Operations. d is the size of matrix dimensions and "$-$" denotes that the operation is not used. The rows corresponding to our techniques are shown in bold face.

	Addition	Multiplication	Rotation	Scalar-multiply
OuterProduct	$O(d \log d)$	$O(d)^*$	$O(d \log d)$	$O(d)^*$
OuterProduct1D	$-$	$\boldsymbol{O(1)^*}$	$-$	$-$
MatVecMul	$O(d \log d)$	$O(d)^*$	$O(d \log d)$	$O(d)^*$
MatVecMul1D	$\boldsymbol{O(\log d)}$	$\boldsymbol{O(1)^*}$	$\boldsymbol{O(\log d)}$	$-$
SwitchOrder1D	$\boldsymbol{O(\log d)}$	$-$	$\boldsymbol{O(\log d)}$	$\boldsymbol{O(1)^*}$
MatMul	$O(d^2 \log d)$	$O(d^2)^*$	$O(d^2 \log d)$	$O(d^2)^*$
MatMul1D	$\boldsymbol{O(d \log d)}$	$\boldsymbol{O(d)^*}$	$\boldsymbol{O(d \log d)}$	$\boldsymbol{O(d)^*}$

$*$: represents that the depth complexity of the operation is $O(1)$

4.2.3 Comparison

In this subsection, we compare the complexities of the techniques described in Section 4.2.1 and Section 4.2.2, in terms of homomorphic operations. For details on effect of these operations on noise and their running time, we refer the readers to [45, Table 1]. In Table 4.1, we have summarized the complexities for all the matrix operations. It can be easily observed that for all the matrix operations, our method has decreased the complexity by a factor of matrix dimension i.e. d.

4.3 Secure Computation in The Cloud

Having defined the matrix primitives in the previous section, we are now ready to proceed to defining protocols for secure computation in the cloud. We showed in the previous section that our method for matrix operations can result in significant performance improvement. Therefore, we adapt the PrivatePCA and PrivateLR protocols proposed in [57] to work with our method for secure matrix operations. The protocols used are basically the same, except we have used our method for matrix operations. Before we proceed with their description, we first describe the input data that will be computed upon:

- We perform statistical procedures on datasets with entries in the real domain. Since the BGV scheme only works on integers, we first need to convert all the real

attributes of a dataset into integers with fixed point precision. Given $x \in \mathbb{R}$, we scale it up by the magnifying constant M and then round it to the nearest integer i.e. $\lfloor Mx \rceil$. For the rest of description, we assume that we are computing statistics on datasets with integer attributes.

- Our setup assumes that the data rows could be coming from different sources that require that their data remains secure from other sources and the cloud. To keep the data secure, each source can encrypt their data rows separately, and send them to the cloud. But there is a problem that given the dataset $(\mathbf{X}, \mathbf{y}) = \{\mathbf{x}_i^\mathsf{T}, y_i\}_{i=0}^{N-1} \in \mathbb{Z}_t^{N \times d}$, it will be too inefficient to compute $\mathbf{X}^\mathsf{T}\mathbf{X}$ or $\mathbf{X}^\mathsf{T}\mathbf{y}$ homomorphically for any practical N. To overcome this problem, we use the fact that $\mathbf{X}^\mathsf{T}\mathbf{X} = \sum_{i=0}^{N-1} \mathbf{x}_i\mathbf{x}_i^\mathsf{T}$ and $\mathbf{X}^\mathsf{T}\mathbf{y} = \sum_{i=0}^{N-1} y_i\mathbf{x}_i^\mathsf{T}$. Therefore, each source can encrypt the matrix $\mathbf{x}_i\mathbf{x}_i^\mathsf{T}$ and the vector $y_i\mathbf{x}_i^\mathsf{T}$ for each data row independent of other sources, which can then be added in the cloud to reconstruct $\mathbf{X}^\mathsf{T}\mathbf{X}$ and $\mathbf{X}^\mathsf{T}\mathbf{y}$.

Remark 23. *Since we use the protocols proposed by Lu et al., except for underlying matrix operations, we defer the security analysis of the protocols to [57, Section VI].*

4.3.1 PrivatePCA

In this subsection, we want to compute the principal components of the data matrix $\mathbf{X} \in \mathbb{Z}_t^{N \times d}$ securely in the cloud, which requires to compute the covariance matrix $\mathbf{\Sigma} = \frac{1}{N}\mathbf{X}^\mathsf{T}\mathbf{X} - \boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T}$. Since we can not perform division while computing $\boldsymbol{\mu}^\mathsf{T}$ and $\frac{1}{N}\mathbf{X}^\mathsf{T}\mathbf{X}$ in the cloud, we rather calculate $N^2\mathbf{\Sigma}$, multiplying $\mathbf{X}^\mathsf{T}\mathbf{X}$ by N and computing $N\boldsymbol{\mu}^\mathsf{T}$ in place of $\boldsymbol{\mu}^\mathsf{T}$. This works because scaling a matrix does not change the direction of its eigen-vectors, it only scales their associated eigen-values. To compute $N^2\boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T}$, we first compute $N\boldsymbol{\mu}^\mathsf{T}$ in row-wise packing, use SwitchOrder1D procedure to get $N\boldsymbol{\mu}$ in column-wise packing, and then input both the ciphertexts into OuterProduct1D to get an encrypted matrix for $N^2\boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T}$. Having computed $N^2\mathbf{\Sigma}$, all that is left is to multiply the vector $\mathbf{v}$ with matrix $N^2\mathbf{\Sigma}$ repeatedly. Since $N^2\mathbf{\Sigma}$ is a symmetric matrix, we can do it directly with MatVecMul1D procedure without the need of SwitchOrder1D procedure. The description of the protocol is given in Algorithm 10.

Algorithm 10 $\mathsf{PrivatePCA}(\{\mathsf{ct}_1(\mathbf{x}_i^\mathsf{T}), \mathsf{ct}(\mathbf{x}_i\mathbf{x}_i^\mathsf{T})\}_{i=0}^{N-1}, T)$

Input:

- $\mathsf{ct}_1(\mathbf{x}_i^\mathsf{T})$: row-wise encryption of i-th row of $\mathbf{X}$
- $\mathsf{ct}(\mathbf{x}_i\mathbf{x}_i^\mathsf{T})$: ciphertext of outer-product of $\mathbf{x}_i^\mathsf{T}$ with itself
- T : number of iterations

Output:

- $\mathbf{u}_1, \lambda_1$: first principal component of $\mathbf{X}$ and its magnitude

Cloud:

$\mathsf{ct}_1(N\boldsymbol{\mu}^\mathsf{T}) = \sum_{i=0}^{N-1} \mathsf{ct}_1(\mathbf{x}_i^\mathsf{T})$ ▷ $N\boldsymbol{\mu}^\mathsf{T} \in \mathbb{Z}_t^d$
$\mathsf{ct}_2(N\boldsymbol{\mu}) = \mathsf{SwitchOrder1D}(\mathsf{ct}_1(N\boldsymbol{\mu}^\mathsf{T}))$
$\mathsf{ct}(N^2\boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T}) = \mathsf{OuterProduct1D}(\mathsf{ct}_2(N\boldsymbol{\mu}), \mathsf{ct}_1(N\boldsymbol{\mu}^\mathsf{T}))$
$\mathsf{ct}(N^2\boldsymbol{\Sigma}) = N \cdot \sum_{i=0}^{N-1} \mathsf{ct}(\mathbf{x}_i\mathbf{x}_i^\mathsf{T}) - \mathsf{ct}(N^2\boldsymbol{\mu}\boldsymbol{\mu}^\mathsf{T})$ ▷ $N^2\boldsymbol{\Sigma} \in \mathbb{Z}_t^{d\times d}$
for $i = 1$ to T **do**
  if i is odd **then**
    $\mathsf{ct}_2(\mathbf{v}^{(i)}) = \mathsf{MatVecMul1D}(\mathsf{ct}(N^2\boldsymbol{\Sigma}), \mathsf{ct}_1(\mathbf{v}^{(i-1)}))$
  else if i is even **then**
    $\mathsf{ct}_1(\mathbf{v}^{(i)}) = \mathsf{MatVecMul1D}(\mathsf{ct}(N^2\boldsymbol{\Sigma}), \mathsf{ct}_2(\mathbf{v}^{(i-1)}))$
  end if
end for
if T is odd **then**
  return $\mathsf{ct}_2(\mathbf{v}^{(T)})$ and $\mathsf{ct}_1(\mathbf{v}^{(T-1)})$
else if T is even **then**
  return $\mathsf{ct}_1(\mathbf{v}^{(T)})$ and $\mathsf{ct}_2(\mathbf{v}^{(T-1)})$
end if

Decryptor:

return $\mathbf{u}_1 = \mathbf{v}^{(T)}/\|\mathbf{v}^{(T)}\|$ and $\lambda_1 = \|\mathbf{v}^{(T)}\|/(N^2 \cdot \|\mathbf{v}^{(T-1)}\|)$

In this way, we can securely compute the *first* principal component of data matrix $\mathbf{X}$. Apart from the PowerMethod that we have in place in the form of PrivatePCA, we need the EigenShift procedure to compute other principal components securely. The EigenShift procedure is easily performed in the cloud through simple operations like matrix addition and homomorphic multiplication, along with outer product on $\mathbf{u}_i$'s which can be done exactly like we did on $N\boldsymbol{\mu}$.

Remark 24. *If the input data is known to be normalised beforehand, we can skip the computation of* $N^2\boldsymbol{\mu}^\mathsf{T}\boldsymbol{\mu}$ *since it will always be zero. Moreover, we do not need to multiply* $\mathbf{X}^\mathsf{T}\mathbf{X}$ *with* N *then.*

4.3.2 PrivateLR

To perform Linear Regression on the dataset $(\mathbf{X}, \mathbf{y}) = \{\mathbf{x}_i^\mathsf{T}, y_i\}_{i=0}^{N-1} \in \mathbb{Z}_t^{N\times d}$, we need to compute $\mathbf{w} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}$. Since we can construct encrypted matrices $\mathbf{X}^\mathsf{T}\mathbf{X}$ and $\mathbf{X}^\mathsf{T}\mathbf{y}$ in the cloud by adding the inputs $\mathbf{x}_i\mathbf{x}_i^\mathsf{T}$ and $y_i\mathbf{x}_i^\mathsf{T}$ respectively, all we need is to invert the matrix $\mathbf{X}^\mathsf{T}\mathbf{X}$ and multiply it with $\mathbf{X}^\mathsf{T}\mathbf{y}$. We use the procedure defined in Algorithm 3 for matrix inversion, that involves just matrix multiplications and additions. Hence, we can use our proposed matrix operations to compute Linear Regression securely. We also need the dominant eigen value λ of matrix $\mathbf{X}^\mathsf{T}\mathbf{X} \in \mathbb{Z}_t^{(d-1)\times(d-1)}$ for stable geometric convergence, which we can get through the PCA protocol before performing Linear Regression. For the scalar $\lambda \in \mathbb{Z}_t$, we use the notation $\mathsf{ct}(\lambda)$ to refer to the ciphertext of a plaintext $\mathbf{V}$ with λ packed in each plaintext slot i.e. $\mathbf{V}[i, j] = \lambda$. The PrivateLR protocol is described in Algorithm 11.

Remark 25. *We have described the protocols as if each data row is coming from an independent source. In practice, we will have multiple data rows coming from a particular source. The sources can send just one ciphertext each for* $\mathbf{x}_i^\mathsf{T}, \mathbf{x}_i\mathbf{x}_i^\mathsf{T}$ *and* $y_i\mathbf{x}_i^\mathsf{T}$ *by performing addition corresponding to their data rows in the plaintext.*

4.4 Choosing The Right Context

The right choice of context parameters can significantly improve the performance of our protocols. Choosing the right parameters is a non-trivial task in `HElib` and in most works using `HElib`, the authors do not describe how they chose the optimal parameters for their application. In this section, we describe the theory used to choose optimal parameters for our setup. Moreover, we found that the parameter choice in [57] is non-optimal. So, we describe how to choose the right parameters for their setup as well.

As described in Section 2.3.3, $\{f_1, \ldots, f_n\}$ is the generating set of $\mathbb{Z}_m^*/\langle t\rangle$. Let the order of f_i in $\mathbb{Z}_m^*/\langle t, f_1, \ldots, f_{i-1}\rangle$ be m_i. Therefore, we have the number of plaintext slots $\ell = |\mathbb{Z}_m^*/\langle t\rangle| = \prod_{i=1}^n m_i$. For efficient data movement operations, we basically have two main requirements:

Algorithm 11 $\mathsf{PrivateLR}(\{\mathsf{ct}_1(y_i\mathbf{x}_i^\mathsf{T}), \mathsf{ct}(\mathbf{x}_i\mathbf{x}_i^\mathsf{T})\}_{i=0}^{N-1}, \mathsf{ct}(\lambda), T)$

Input:

- $\mathsf{ct}_1(y_i\mathbf{x}_i^\mathsf{T})$: row-wise encryption of i-th row of $\mathbf{X}$ multiplied with i-th element of $\mathbf{y}$
- $\mathsf{ct}(\mathbf{x}_i\mathbf{x}_i^\mathsf{T})$: ciphertext of outer-product of $\mathbf{x}_i^\mathsf{T}$ with itself
- $\mathsf{ct}(\lambda)$: ciphertext of dominant eigen-value of $\mathbf{X}^\mathsf{T}\mathbf{X}$
- T : number of iterations

Output:

- $\mathbf{w}$: weight vector $\mathbf{w} \in \mathbb{Z}_t^{d-1}$

Cloud:

$\mathsf{ct}_1(\mathbf{X}^\mathsf{T}\mathbf{y}) = \sum_{i=0}^{N-1} \mathsf{ct}_1(y_i\mathbf{x}_i^\mathsf{T})$ ▷ $\mathbf{X}^\mathsf{T}\mathbf{y} \in \mathbb{Z}_t^{(d-1)}$

$\mathsf{ct}(\mathbf{A}^0) = \mathsf{ct}(\mathbf{X}^\mathsf{T}\mathbf{X}) = \sum_{i=0}^{N-1} \mathsf{ct}(\mathbf{x}_i\mathbf{x}_i^\mathsf{T})$ ▷ $\mathbf{X}^\mathsf{T}\mathbf{X} \in \mathbb{Z}_t^{(d-1)\times(d-1)}$

$\mathsf{ct}(\mathbf{R}^0) = \mathsf{ct}(\mathbf{I}),\ \mathsf{ct}(\alpha^{(0)}) = \mathsf{ct}(\lambda)$

for $i = 1$ to T **do**

  $\mathsf{ct}(\mathbf{B}) = 2 \cdot \mathsf{ct}(\alpha^{(i-1)}) \cdot \mathbf{I} - \mathsf{ct}(\mathbf{A}^{(i-1)})$

  $\mathsf{ct}(\mathbf{R}^{(i)}) = \mathsf{MatMul1D}(\mathsf{ct}(\mathbf{R}^{(i-1)}), \mathsf{ct}(\mathbf{B}))$

  $\mathsf{ct}(\mathbf{A}^{(i)}) = \mathsf{MatMul1D}(\mathsf{ct}(\mathbf{A}^{(i-1)}), \mathsf{ct}(\mathbf{B}))$

  $\mathsf{ct}(\alpha^{(i)}) = \mathsf{ct}(\alpha^{(i-1)}) \cdot \mathsf{ct}(\alpha^{(i-1)})$

end for ▷ $\mathsf{ct}(\lambda^{2^T}(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}) = \mathsf{ct}(\mathbf{R}^{(T)})$

$\mathsf{ct}_2(\lambda^{2^T}\mathbf{w}) = \mathsf{MatVecMul1D}(\mathsf{ct}(\lambda^{2^T}(\mathbf{X}^T\mathbf{X})^{-1}), \mathsf{ct}_1(\mathbf{X}^\mathsf{T}\mathbf{y}))$

return $\mathsf{ct}_2(\lambda^{2^T}\mathbf{w})$ and $\mathsf{ct}(\lambda^{2^T})$ ▷ $\mathsf{ct}(\alpha^{(T)}) = \mathsf{ct}(\lambda^{2^T})$

Decryptor:

return $\mathbf{w}$ from $\lambda^{2^T}\mathbf{w}$ by dividing with λ^{2^T}.

1. For each $i \in [n]$, we should have $\mathsf{ord}_{\mathbb{Z}_m^*}(f_i) = m_i$, since it allows true rotations on slots.

2. ℓ should be kept close to the number of plaintext slots required by the application.

For more details on why we have these requirements, refer [44, Section 4]. Before proceeding to the solution that satisfies the requirements mentioned above, we derive some results on which our solution will be based.

Theorem 26. *Let G be a finite abelian group. Suppose we have an element $g \in G$ and a subgroup N such that $\mathsf{ord}(g) = k$ and $|N| = K$. Then the following holds:*

$$\gcd(k, K) = 1 \implies \mathsf{ord}_G(g) = \mathsf{ord}_{G/N}(g).$$

Proof. It is clear from the definition of a quotient group that:

$$\mathsf{ord}_G(g) = \mathsf{ord}_{G/N}(g) \iff \langle g \rangle \cap N = \{e\},$$

where e is the identity element of G. Therefore, to prove the theorem, it is sufficient to prove that $\gcd(k, K) = 1$ implies $\langle g \rangle \cap N = \{e\}$. Let g' be an arbitrary element $\in \langle g \rangle \cap N$. This implies that $\mathsf{ord}(g')|k$ as well as $\mathsf{ord}(g')|K$. Since $\gcd(k, K) = 1$, g' has order 1, implying $g' = e$. Since we assumed g' to be a general element in $\langle g \rangle \cap N$, we have $\langle g \rangle \cap N = \{e\}$. □

Corollary 27. *Let $g \in G$ be an element of a finite abelian group G with $\mathsf{ord}(g) = k$. Suppose we have n elements $g_1, \ldots, g_n \in G$ with $\mathsf{ord}(g_i) = k_i$. We denote the subgroup generated by the set $\{g_1, \ldots, g_n\}$ with N i.e. $N = \langle g_1, \ldots, g_n \rangle$. Then the following holds:*

$$\forall i \in [n], \gcd(k_i, k) = 1 \implies \mathsf{ord}_G(g) = \mathsf{ord}_{G/N}(g)$$

Proof. It is a well-known equality that given subgroups $N_1, N_2 \triangleleft G$, we have a subgroup $N_1N_2 = \{h_1h_2 | h_1 \in N_1, h_2 \in N_2\}$ such that $|N_1N_2| = |N_1| \cdot |N_2|/|N_1 \cap N_2|$. Therefore, we have a subgroup N generated by $\{g_i\}_{i=1}^n$ of order K which divides the product of all k_i's i.e. $K|(\prod_{i=1}^n k_i)$. Since $\forall i \in [n], \gcd(k_i, k) = 1$, we have $\gcd(k, K) = 1$. Hence, by Theorem 26, the proof is complete. □

4.4.1 Lu et al.'s Method

For our application using Lu et al.'s method, we need a one-dimensional array of size m_1, where m_1 is greater than and close to d. Moreover, this dimension should be a *good* dimension. These two conditions satisfy our above-mentioned requirements for efficient data movement. We achieve these conditions by choosing m and t such that:

1. $m = k \cdot m_1 + 1 \in \mathbb{P}$,
2. $\gcd(k, m_1) = 1$,
3. $\mathsf{ord}(t) = k$.

Since m is chosen to be a prime, $\mathbb{Z}_m^*$ is a cyclic group of order $m_1 \cdot k$. This ensures that it has an element f_1 of order m_1. If we choose an element of order k (exists because $\mathbb{Z}_m^*$ is cyclic) as t, then the number of plaintext slots $\ell = |\mathbb{Z}_m^*/\langle t \rangle| = m_1$. Given $\gcd(k, m_1) = 1$, by Corollary 27, we get $\mathsf{ord}_{\mathbb{Z}_m^*}(f_1) = \mathsf{ord}_{\mathbb{Z}_m^*/\langle t \rangle}(f_1) = m_1$. Therefore, $\mathbb{Z}_m^*/\langle t \rangle = \langle f_1 \rangle$ and we get a one-dimensional array with a *good* dimension of size m_1.

4.4.2 Our Method

The optimal parameters for our method lead to a *two*-dimensional hypercube with *good* dimensions of size m_1 and m_2, where $m_1 = d$ and m_2 is greater than and close to d. These parameters satisfy the requirements for efficiency, and are achieved by choosing m and t such that:

1. $m = k \cdot m_1 \cdot m_2 + 1 \in \mathbb{P}$,
2. $\gcd(k, m_1) = \gcd(k, m_2) = \gcd(m_1, m_2) = 1$,
3. $\mathsf{ord}(t) = k$.

Since $\mathbb{Z}_m^*$ is a cyclic group of order $k \cdot m_1 \cdot m_2$, we can find f_1 and f_2 in $\mathbb{Z}_m^*$ with order m_1 and m_2 respectively. For a chosen t of order k, we have $\ell = |\mathbb{Z}_m^*/\langle t \rangle| = m_1 \cdot m_2$. Given $\gcd(k, m_1) = 1$ and $\gcd(k, m_2) = \gcd(m_1, m_2) = 1$, by Corollary 27, we get $\mathsf{ord}_{\mathbb{Z}_m^*}(f_1) = \mathsf{ord}_{\mathbb{Z}_m^*/\langle t \rangle}(f_1) = m_1$ and $\mathsf{ord}_{\mathbb{Z}_m^*}(f_2) = \mathsf{ord}_{\mathbb{Z}_m^*/\langle t, f_1 \rangle}(f_2) = m_2$ respectively. Therefore, $\mathbb{Z}_m^*/\langle t \rangle = \langle f_1, f_2 \rangle$ and we get a *two*-dimensional hypercube with *good* dimensions of size m_1 and m_2.

Remark 28. *It gets relatively easier to find an optimal m for Lu et al.'s method as the matrix dimension grows. Therefore, for a considerably large d, we have to settle with relaxed parameters for our method. Despite the relaxed parameters, our method still vastly outperforms Lu et al. 's because our method improves the complexity of their techniques by a factor of d (see Table 4.1).*

4.4.3 CRT Method for Plaintext Modulus

In `HElib`, the maximum plaintext precision is 60 bits, which is not sufficient for our applications. To address this problem, we split the plaintext modulus t into e different

primes t_i each of bitsize < 60 bits such that $t = \prod_{i=1}^{e} t_i$ for some $e \in \mathbb{Z}$. Now, we create e different instances of the cryptosystem, with i-th instance operating on plaintext modulus t_i, and encrypt our inputs in each one of them. Then, the protocol is performed for every instance and the decryption results are combined using the Chinese Remainder Theorem to get plaintext precision of $\prod_{i=1}^{e} \lfloor \log_2(t_i) \rfloor$ bits. Using the CRT method, we essentially have e ciphertexts as opposed to one. This does not pose a problem since the CRT method is completely parallelizable. Moreover, increasing e leads to a smaller t_i, and as a result, lesser number of levels L. Therefore, we get a more efficient implementation, given we have enough CPU cores.

Remark 29. *In practice, we want to choose each t_i of p-bits, and to maintain security, m is chosen to be greater than a lower bound which depends on t and the number of levels L. To choose each t_i, we search the equivalence class of elements with order k for distinct p-bit members. The number m is chosen such that m_1 and m_2 are close to d while keeping m as close to the lower bound as possible.*

4.5 Experimental Details and Results

We implemented the protocols defined in Section 4.3 using both our method and Lu et al.'s method. The implementations were written in C++ and compiled with g++ 4.8.5. We used `HElib` [44] for the implementation of the BGV scheme. Our experiments ran on a machine with Intel(R) Xeon(R) E5-4650@2.70GHz processor and 1TB memory running CentOS 7.5.1804. In all our experiments, we use CRT method with $e = 8$ (see Section 4.4.3), and have parallelized the implementations over 8 threads.

4.5.1 Experimental Setup

Our experimental setup is identical to the one used in [57]. We have conducted our experiments on five datasets from the UCI Machine Learning Repository [25]. For the analysis of the effect of number of iterations and different magnification constants on performance, experiments were performed on the *adult* dataset that has $N = 32561$ records with $d = 6$ numerical attributes each. More specifically, we took iteration numbers

$T = \{3, 4, 5\}$ for PrivatePCA protocol and iteration numbers $T = \{1, 2, 3\}$ for PrivateLR protocol. Three magnification constants $M = \{10, 100, 1000\}$ were considered for both the protocols. Experiments on the other datasets serve to show the scalability of our approach and we have taken $T = 5, M = 1000$ and $T = 3, M = 1000$ for PrivatePCA and PrivateLR protocol, respectively on these datasets. Following the experimental setup of [57], we have also performed all our experiments on normalized datasets, with the assumption that the data is coming from a single source. Since our experimental setup is identical to Lu et al.'s, we defer to [57, Appendix A] for discussion on iteration-error tradeoffs.

4.5.2 Selection of Parameters

Our parameter selection basically involves choosing m and t_i that define the plaintext spaces $\mathbb{Z}[X]/(\Phi_m(X), t_i)$ for $1 \leq i \leq e$, and L which defines the number of levels. Given a dataset with each entry bounded by B, we get a lower bound on the bitsize p of t_i from the inequality $t_i > (B^2MNd)^{T/8}$ for the PrivatePCA protocol and $t_i > M^{1/8}(B^2MNd)^{2^T/8}$ for the PrivateLR protocol. An ephemeral p-bit prime following this inequality is sampled and a dry run is performed to find the required number of levels L.

According to the analysis of [37, Appendix C.3], we get κ-bit security from the BGV scheme if the following inequality holds:

$$\phi(m) > \frac{(L(\log \phi(m) + 23) - 8.5)(\kappa + 110)}{7.2}.$$

Since we only choose a prime m in our setup, we can replace $\phi(m)$ with $m - 1$ in the above inequality. Given L (from dry run) and a minimum security requirement κ, from the above inequality, we get a lower bound on m. Given the lower bound on m and bitsize of t_i, we choose m and the t_i's according to the method described in Section 4.4 for both our method and Lu et al.'s. In the experiments concerning PrivatePCA, we have taken $m_1 = d$ for Lu et al.'s method and $(m_1, m_2) = (d, d + 1)$ for our method. For experiments concerning PrivateLR, we have taken $m_1 = d - 1$ for Lu et al.'s method and $(m_1, m_2) = (d - 1, d)$ for our method.

Remark 30. *We have chosen the best possible plaintext dimensions for our experiments. We could easily find such parameters since the matrix dimension was not too large (≤ 20).*

For operations on matrices with large dimensions, we can relax the upper bound on the size of plaintext dimensions, or we can use the block matrix multiplication method.

For other parameters in `HElib`, we use the default values such as $\sigma = 3.2$ (standard deviation parameter for error distribution), $H = 64$ (Hamming weight of secret key), $r = 1$ (lifting factor) and $c = 3$ (number of columns in key switching matrix). For all the experiments, we have maintained a minimum security of 128 bits.

4.5.3 Results and Analysis

In Table 4.2 and Table 4.3, we have compared the performance of our method with Lu et al.'s method for performing PrivatePCA and PrivateLR respectively. For PrivatePCA, we have only considered the time taken to compute the first principal component, and for PrivateLR, we exclude the time taken to compute the first eigen value λ_1. From the results of Table 4.2 and Table 4.3, the following observations can be made:

- *Encryption Time*: The encryption time for our method is d times faster than Lu et al.'s on an average because we require *one* ciphertext to encrypt a matrix, as opposed to d ciphertexts for their method.
- *Evaluation Time*: The difference in homomorphic evaluation time is in accordance with the complexities of matrix operations described in Table 4.1. As expected, the evaluation time of our method is d times faster than Lu et al.'s.
- *Decryption Time*: Since the output of both the protocols is a vector, our method outputs the same number of ciphertexts as Lu et al.'s method. Our decryption time is more because the decoding operation takes more time for our method, given we are using more plaintext slots.
- *Depth Requirement*: Our method has comparatively less depth requirement. This is due to the fact that the MatVecMul procedure requires an additional scalar-multiplication operation, which adds moderate noise (see [45, Table 1]), compared to the MatVecMul1D procedure. This effect is more pronounced in PrivatePCA since it involves the matrix-vector multiplication in each iteration, while PrivateLR involves it only once.

Table 4.2 Performance comparison for the PrivatePCA protocol of our method (Section 4.3.1) with Lu et al.'s method ([57, Section V.C], with our optimized parameters) to find the first principal component. For all the experiments, we split the plaintext modulus into 8 primes $t_1, \ldots, t_8$ using our CRT method and use 8 threads to execute them all in parallel. The performance for our method is shown in bold face.

Dataset details			**Exp. Settings**		**BGV settings**			**Performance (seconds)**			
Dataset	N	d	M	T	$\lfloor \log_2(t_i) \rfloor$	m	L	Encryption	Hom. Eval.	Decryption	Total time
adult	32561	6	10	3	11	12703	11	0.4788	5.4683	0.1319	6.079
						11047	**9**	**0.0912**	**0.8562**	**0.2713**	**1.2187**
				4	12	14503	13	0.5767	8.8194	0.1363	9.5324
						13063	**11**	**0.1161**	**1.529**	**0.2893**	**1.9344**
				5	15	18583	17	1.2227	22.8318	0.2323	24.2868
						15331	**13**	**0.1469**	**2.2915**	**0.5591**	**2.9975**
			100	3	11	12703	11	0.4788	5.4683	0.1319	6.079
						11047	**9**	**0.0912**	**0.8562**	**0.2713**	**1.2187**
				4	14	14683	13	0.5288	9.0437	0.1364	9.7089
						14071	**11**	**0.1199**	**1.7285**	**0.296**	**2.1444**
				5	17	23011	21	1.4851	30.8704	0.2728	32.6283
						15331	**13**	**0.1423**	**2.3695**	**0.5595**	**3.0713**
			1000	3	11	12703	11	0.4788	5.4683	0.1319	6.079
						11047	**9**	**0.0912**	**0.8562**	**0.2713**	**1.2187**
				4	15	14683	13	0.6396	8.9445	0.2	9.7841
						14071	**11**	**0.1328**	**1.7843**	**1.1814**	**3.0985**
				5	19	25603	23	1.6289	33.835	0.4118	35.8757
						19447	**17**	**0.2769**	**5.5374**	**0.6144**	**6.4287**
auto-mpg	398	7	1000	5	15	19069	17	1.3713	33.1703	0.2885	34.8301
						15401	**13**	**0.1583**	**2.5215**	**1.3696**	**4.0494**
winequality	4898	12	1000	5	18	26293	23	3.4114	84.1018	0.5672	88.0804
						19501	**15**	**0.2865**	**5.3242**	**2.2644**	**7.8751**
forestfires	517	13	1000	5	16	20749	19	3.074	82.9019	0.4014	86.3773
						15107	**13**	**0.1651**	**2.9428**	**1.4105**	**4.5184**
communities	1994	20	1000	5	17	22741	21	5.1348	148.784	0.6661	154.5849
						18061	**15**	**0.2803**	**6.3988**	**2.09**	**8.7691**

Table 4.3 Performance comparison for the PrivateLR protocol of our method (Section 4.3.2) with Lu et al.'s method ([57, Section V.C], with our optimized parameters). We excluded the computational time required to compute the largest eigen-value λ_1. For all the experiments, we split the plaintext modulus into 8 primes $t_1, \ldots, t_8$ using our CRT method and use 8 threads to execute them all in parallel. The performance for our method is shown in bold face.

Dataset details			**Exp. settings**		**BGV settings**			**Performance (Seconds)**			
Dataset	N	d	M	T	$\lfloor \log_2(t_i) \rfloor$	m	L	Encryption	Hom. Eval.	Decryption	Total time
adult	32561	6	10	1	11	10531	9	0.4964	3.8267	0.0733	4.3964
						8311	**7**	**0.1959**	**0.6444**	**0.1471**	**0.9874**
				2	15	13121	11	0.5524	28.1256	0.1198	28.7978
						10771	**9**	**0.2719**	**4.7969**	**0.2767**	**5.3455**
				3	26	23741	21	1.8307	159.8920	0.2256	161.9483
						20731	**19**	**0.7658**	**34.0543**	**0.6125**	**35.4326**
			100	1	11	10531	9	0.4964	3.8267	0.0733	4.3964
						8311	**7**	**0.1959**	**0.6444**	**0.1471**	**0.9874**
				2	17	15031	13	0.6564	33.8865	0.1153	34.6582
						14011	**11**	**0.2912**	**6.5338**	**0.3211**	**7.1461**
				3	30	27011	23	1.8077	188.1300	0.2712	190.2089
						23011	**21**	**0.7817**	**38.3839**	**0.6418**	**39.8074**
			1000	1	11	10531	9	0.4964	3.8267	0.0733	4.3964
						8311	**7**	**0.1959**	**0.6444**	**0.1471**	**0.9874**
				2	19	17011	15	1.2712	60.1418	0.1906	61.6036
						14731	**13**	**0.3093**	**7.4470**	**0.3506**	**8.1069**
				3	33	28111	25	2.0766	206.4470	0.2745	208.7981
						26431	**23**	**0.9210**	**45.4241**	**0.6954**	**47.0405**
auto-mpg	398	7	1000	3	26	24007	21	2.0630	228.9190	0.2190	231.2010
						21211	**19**	**0.8031**	**49.1406**	**0.6096**	**50.5533**
winequality	4898	12	1000	3	30	26203	23	3.5929	1169.4800	0.3569	1173.4298
						23629	**21**	**0.8884**	**95.7648**	**3.1740**	**99.8228**
forestfires	517	13	1000	3	27	25117	23	3.8555	1158.9700	0.3843	1163.2098
						22621	**21**	**0.8640**	**122.2040**	**2.9421**	**126.0101**
communities	1994	20	1000	3	29	27361	23	6.0348	4399.6500	0.4428	4406.1276
						26981	**21**	**0.9311**	**207.5400**	**4.2580**	**212.7291**

Chapter 5

Conclusion and Future Work

In this chapter, we give the conclusion and discuss the future works.

5.1 Conclusion

We propose a new method to pack a matrix into a single ciphertext so that it also enables efficient matrix multiplication over the packed ciphertexts. Our packing method generalizes Yasuda et al.'s methods (Security Comm. Networks 2015 and ACISP 2015), which are for secure inner product. We generalized our packing method [28] for secure *multiple* matrix multiplications $\mathbf{A}_1 \times \cdots \times \mathbf{A}_\ell$ with integer entries over the BV and the BGV scheme. Our packing method can be used over other ring-based SHE schemes such as the FV [30], YASHE [7], and NTRU [56] schemes.

We also give some modifications to make our packing method practical for matrices with large size and entries. Over the BV scheme, our non-optimal implementation took about 12 seconds (resp., 15.6 minutes) for secure multiplications between two matrices (resp., among three matrices) with 64×64 size and 64-bit entries (see Tables 3.1 and 3.2 for our implementation results). On the other hand, 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. 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 an 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.

We also made the protocols for PCA and Linear Regression, proposed in [57], more efficient by improving their underlying matrix operations by a factor of data dimension. This was achieved by utilising the hypercube structure of the plaintext slots. Our techniques for matrix operations are flexible and can be used for any application involving matrix operations. In addition to this, we show how to choose optimal parameters for our method as well as for the method proposed in [57] by Lu et al. With our improved matrix procedures, we reduced the evaluation time of largest case ($d = 20, N = 1994, M = 1000, T = \{5\}$) for PCA from 149 seconds to 6.5 seconds, and of largest case ($d = 20, N = 1994, M = 1000, T = \{3\}$) for Linear Regression from 4400 seconds to 207 seconds. With this work, we show that the plaintext space can be utilised to a larger extent, leading to much faster PCA and Linear Regression computation.

5.2 Future Work

Since our packing method is different for each matrix 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. We can not extract the targeted coefficient after multiplication. Therefore, our parameter grows exponentially (for example $n \geq m^{\ell+1}$, where n, m and ℓ is the lattice dimension, the dimension of the matrix, and the number of matrices being multiplied). One can extract targeted coefficient from encrypted polynomial after each multiplication using the method proposed in [58, 48]. Since this extraction is not very efficient and capable of extracting only one coefficient, we will focus on efficient technique for the multiple extractions. After the above improvements, we try to use our method to solve real-world problem securely on encrypted data.

Apart from the above, another future work is the extension of the hypercube structure from *two* to *three*-dimensions. As a result, we can reduce the complexity of matrix multiplication from $O(d \log d)$ to $O(\log d)$, where d is the matrix dimension. This can be

done by constructing a multiplication procedure along the lines of DNS algorithm [24]. A limitation of this approach is that it will be harder for us to find optimal parameters. Future work can explore the situations where it is beneficial to use a higher dimension plaintext structure. In this work, we have considered predictive statistics through *exact* arithmetic and controlled the high plaintext modulus growth by using the CRT method and leveraging multiple cores to limit the impact. Switching to a scheme that allows *approximate* arithmetic, like the one described in [14], can help resolve this problem. Encoding techniques that better utilise the finite field structure of a plaintext slot can also help in containing the large plaintext modulus.

References

[1] Ajtai, M.: Generating Hard Instances of the Short Basis Problem. In: J. Wiedermann, P. van Emde Boas, M. Nielsen (eds.) Automata, Languages and Programming, pp. 1–9. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)

[2] Alwen, J., Peikert, C.: Generating Shorter Bases for Hard Random Lattices. Theory of Computing Systems **48**(3), 535–553 (2011). DOI 10.1007/s00224-010-9278-3. URL https://doi.org/10.1007/s00224-010-9278-3

[3] Benaloh, J.D.C.: Verifiable Secret-ballot Elections. Ph.D. thesis, New Haven, CT, USA (1987). AAI8809191

[4] Boneh, D.: The Decision Diffie-Hellman problem. In: J.P. Buhler (ed.) Algorithmic Number Theory, pp. 48–63. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)

[5] Boneh, D., Gentry, C., Halevi, S., Wang, F., Wu, D.J.: Private database queries using somewhat homomorphic encryption. In: Applied Cryptography and Network Security–ACNS 2013, *Lecture Notes in Computer Science*, vol. 7954, pp. 102–118. Springer (2013)

[6] Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Theory of Cryptography–TCC 2005, *Lecture Notes in Computer Science*, vol. 3378, pp. 325–341. Springer (2005)

[7] Bos, J.W., Lauter, K., Loftus, J., Naehrig, M.: Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In: Cryptography and Coding–IMACC 2013, *Lecture Notes in Computer Science*, vol. 8308, pp. 45–64. Springer (2013)

[8] Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine Learning Classification over Encrypted Data. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015 (2015). URL https://www.ndss-symposium.org/ndss2015/machine-learning-classification-over-encrypted-data

[9] Brakerski, Z.: Fully Homomorphic Encryption Without Modulus Switching from Classical GapSVP. In: Proceedings of the 32Nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417, pp. 868–886. Springer-Verlag, Berlin, Heidelberg (2012). DOI 10.1007/978-3-642-32009-5_50. URL https://doi.org/10.1007/978-3-642-32009-5_50

[10] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (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)

[11] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical Hardness of Learning with Errors. In: Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, STOC '13, pp. 575–584. ACM, New York, NY, USA (2013). DOI 10.1145/2488608.2488680. URL http://doi.acm.org/10.1145/2488608.2488680

[12] Brakerski, Z., Vaikuntanathan, V.: Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In: P. Rogaway (ed.) Advances in Cryptology – CRYPTO 2011, pp. 505–524. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[13] Camenisch, J., Lysyanskaya, A.: An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. Cryptology ePrint Archive, Report 2001/019 (2001). https://eprint.iacr.org/2001/019

[14] Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic Encryption for Arithmetic of Approximate Numbers. In: T. Takagi, T. Peyrin (eds.) Advances in Cryptology – ASIACRYPT 2017, pp. 409–437. Springer International Publishing, Cham (2017)

[15] Cheon, J.H., Kim, M., Kim, M.: 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)

[16] Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Advacens in Cryptology–ASIACRYPT 2016, *Lecture Notes in Computer Science*, vol. 10031, pp. 3–33. Springer (2016)

[17] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast Fully Homomorphic Encryption over the Torus. Cryptology ePrint Archive, Report 2018/421 (2018). https://eprint.iacr.org/2018/421

[18] Cohen, J.D., Fischer, M.J.: A Robust and Verifiable Cryptographically Secure Election Scheme. In: Proceedings of the 26th Annual Symposium on Foundations of Computer Science, SFCS '85, pp. 372–382. IEEE Computer Society, Washington, DC, USA (1985). DOI 10.1109/SFCS.1985.2. URL https://doi.org/10.1109/SFCS.1985.2

[19] Costache, A., Smart, N.P.: Which Ring Based Somewhat Homomorphic Encryption Scheme is Best? In: Topics in Cryptology–CT-RSA 2016, *Lecture Notes in Computer Science*, vol. 9610, pp. 325–340. Springer (2016)

[20] Cramer, R., Franklin, M., Schoenmakers, B., Yung, M.: Multi-authority Secret-ballot Elections with Linear Work. In: Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques, EUROCRYPT'96, pp. 72–83. Springer-Verlag, Berlin, Heidelberg (1996). URL http://dl.acm.org/citation.cfm?id=1754495.1754505

[21] Cramer, R., Gennaro, R., Schoenmakers, B.: A Secure and Optimally Efficient Multi-Authority Election Scheme. In: W. Fumy (ed.) Advances in Cryptology — EUROCRYPT '97, pp. 103–118. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[22] Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: H. Krawczyk (ed.) Advances in Cryptology — CRYPTO '98, pp. 13–25. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)

[23] Damgård, I.: Towards Practical Public Key Systems Secure Against Chosen Ciphertext attacks. In: J. Feigenbaum (ed.) Advances in Cryptology — CRYPTO '91, pp. 445–456. Springer Berlin Heidelberg, Berlin, Heidelberg (1992)

[24] Dekel, E., Nassimi, D., Sahni, S.: Parallel Matrix and Graph Algorithms. SIAM Journal on Computing **10**(4), 657–675 (1981). DOI 10.1137/0210049. URL https://doi.org/10.1137/0210049

[25] Dheeru, D., Karra Taniskidou, E.: UCI Machine Learning Repository (2017). URL http://archive.ics.uci.edu/ml

[26] van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully Homomorphic Encryption over the Integers. Cryptology ePrint Archive, Report 2009/616 (2009). https://eprint.iacr.org/2009/616

[27] Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. Microsoft Research (2016)

[28] Duong, D.H., Mishra, P.K., Yasuda, M.: Efficient Secure Matrix Multiplication Over LWE-Based Homomorphic Encryption. Tatra Mountains Mathematical Publications **67**(1) (2016)

[29] El Gamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: Proceedings of CRYPTO 84 on Advances in Cryptology, pp. 10–18. Springer-Verlag New York, Inc., New York, NY, USA (1985). URL http://dl.acm.org/citation.cfm?id=19478.19480

[30] Fan, J., Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, Report 2012/144 (2012). URL https://eprint.iacr.org/2012/144.pdf

[31] Fousse, L., Lafourcade, P., Alnuaimi, M.: Benaloh's Dense Probabilistic Encryption Revisited. CoRR **abs/1008.2991** (2010). URL http://arxiv.org/abs/1008.2991

[32] Fox, G., Otto, S., Hey, A.: Matrix algorithms on a hypercube I: Matrix multiplication. Parallel Computing **4**(1), 17 – 31 (1987). DOI https://doi.org/10.1016/0167-8191(87)90060-3. URL http://www.sciencedirect.com/science/article/pii/0167819187900603

[33] Freedman, M.J., Nissim, K., Pinkas, B.: Efficient Private Matching and Set Intersection. In: C. Cachin, J.L. Camenisch (eds.) Advances in Cryptology - EUROCRYPT 2004, pp. 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[34] Galbraith, S.D., Gebregiyorgis, S.W., Murphy, S.: Algorithms for the Approximate Common Divisor Problem. Cryptology ePrint Archive, Report 2016/215 (2016). https://eprint.iacr.org/2016/215

[35] Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Symposium on Theory of Computing–STOC 2009, pp. 169–178. ACM (2009)

[36] Gentry, C., Halevi, S., Smart, N.P.: Fully Homomorphic Encryption with Polylog Overhead. In: Advances in Cryptology – EUROCRYPT 2012, *Lecture Notes in Computer Science*, vol. 7237, pp. 465–482. Springer (2012)

[37] Gentry, C., Halevi, S., Smart, N.P.: Homomorphic Evaluation of the AES Circuit. In: R. Safavi-Naini, R. Canetti (eds.) Advances in Cryptology – CRYPTO 2012, pp. 850–867. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[38] Gentry, C., Halevi, S., Vaikuntanathan, V.: A Simple BGN-Type Cryptosystem from LWE. In: H. Gilbert (ed.) Advances in Cryptology – EUROCRYPT 2010, pp. 506–522. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[39] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for Hard Lattices and New Cryptographic Constructions. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, STOC '08, pp. 197–206. ACM, New York, NY, USA (2008). DOI 10.1145/1374376.1374407. URL http://doi.acm.org/10.1145/1374376.1374407

[40] Goldreich, O., Goldwasser, S., Halevi, S.: Public-key cryptosystems from lattice reduction problems. In: B.S. Kaliski (ed.) Advances in Cryptology — CRYPTO '97, pp. 112–131. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[41] Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences **28**(2), 270 – 299 (1984). DOI https://doi.org/10.1016/0022-0000(84)90070-9. URL http://www.sciencedirect.com/science/article/pii/0022000084900709

[42] Graepel, T., Lauter, K., Naehrig, M.: ML Confidential: Machine Learning on Encrypted Data. In: Proceedings of the 15th International Conference on Information Security and Cryptology, ICISC'12, pp. 1–21. Springer-Verlag, Berlin, Heidelberg (2013). DOI 10.1007/978-3-642-37682-5_1. URL http://dx.doi.org/10.1007/978-3-642-37682-5_1

[43] Guo, C., Higham, N.: A Schur-Newton Method for the Matrix p-th Root and its Inverse. SIAM Journal on Matrix Analysis and Applications **28**(3), 788–804 (2006). DOI 10.1137/050643374. URL https://doi.org/10.1137/050643374

[44] Halevi, S., Shoup, V.: Design and implementation of a homomorphic-encryption library (2013). URL http://people.csail.mit.edu/shaih/pubs/he-library.pdf

[45] Halevi, S., Shoup, V.: Algorithms in HElib. In: J.A. Garay, R. Gennaro (eds.) Advances in Cryptology – CRYPTO 2014, pp. 554–571. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[46] Halevi, S., Shoup, V.: HElib https://github.com/shaih/HElib (2014). URL https://github.com/shaih/HElib

[47] Halevi, S., Shoup, V.: Faster homomorphic linear transformations in HElib. Cryptology ePrint Archive, Report 2018/244 (2018). URL https://eprint.iacr.org/2018/244.pdf

[48] Ishimaki, Y., Yamana, H.: Non-Interactive and Fully Output Expressive Private Comparison. Accepted in Indocrypt-2018

[49] J. Hoffstein D. Lieman, J.P., Silverman, J.: NTRU: A public key cryptosystem. In: in Proc. EUROCRYPT, p. 211–225 (2001)

[50] Jiang, X., Kim, M., Lauter, K., Song, Y.: Secure Outsourced Matrix Computation and Application to Neural Networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, pp. 1209–1222. ACM, New York, NY, USA (2018). DOI 10.1145/3243734.3243837. URL http://doi.acm.org/10.1145/3243734.3243837

[51] Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A Low Latency Framework for Secure Neural Network Inference. Cryptology ePrint Archive, Report 2018/073 (2018). https://eprint.iacr.org/2018/073

[52] Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition, 2nd edn. Chapman & Hall/CRC (2014)

[53] Kushilevitz, E., Ostrovsky, R.: Replication is Not Needed: Single Database, Computationally-private Information Retrieval. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS '97, pp. 364–. IEEE Computer Society, Washington, DC, USA (1997). URL http://dl.acm.org/citation.cfm?id=795663.796363

[54] Lepoint, T., Naehrig, M.: A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In: Progress in Cryptology–AFRICACRYPT 2014, *Lecture Notes in Computer Science*, vol. 8469, pp. 318–335. Springer (2014)

[55] Lindner, R., Peikert, C.: Better Key Sizes (and Attacks) for LWE-Based Encryption. In: A. Kiayias (ed.) Topics in Cryptology – CT-RSA 2011, pp. 319–339. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[56] López-Alt, A., Tromer, E., Vaikuntanathan, V.: 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, pp. 1219–1234. ACM (2012)

[57] Lu, W., Kawasaki, S., Sakuma, J.: 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). URL https://eprint.iacr.org/2016/1163

[58] Lu, W.j., Zhou, J.j., Sakuma, J.: Non-interactive and Output Expressive Private Comparison from Homomorphic Encryption. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIACCS '18, pp. 67–74. ACM, New York, NY, USA (2018). DOI 10.1145/3196494.3196503. URL http://doi.acm.org/10.1145/3196494.3196503

[59] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) **60**(6), 43 (2013)

[60] Mishra, P.K., Duong, D.H., Yasuda, M.: Enhancement for Secure Multiple Matrix Multiplications over Ring-LWE Homomorphic Encryption. In: International Conference on Information Security Practice and Experience–ISPEC 2017, *Lecture Notes in Computer Science*, vol. 10701, pp. 320–330. Springer (2017)

[61] Naccache, D., Stern, J.: A New Public Key Cryptosystem Based on Higher Residues. In: Proceedings of the 5th ACM Conference on Computer and Communications Security, CCS '98, pp. 59–66. ACM, New York, NY, USA (1998). DOI 10.1145/288090.288106. URL http://doi.acm.org/10.1145/288090.288106

[62] Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can Homomorphic Encryption Be Practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop–CCSW 2011, pp. 113–124. ACM (2011)

[63] Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: K. Nyberg (ed.) Advances in Cryptology — EUROCRYPT'98, pp. 308–318. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)

[64] Paar, C., Pelzl, J.: Introduction to Public-Key Cryptography, pp. 149–171. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). DOI 10.1007/978-3-642-04101-3_6. URL https://doi.org/10.1007/978-3-642-04101-3_6

[65] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Advances in Cryptology–EUROCRYPT 1999, *Lecture Notes in Computer Science*, vol. 1592, pp. 223–238 (1999)

[66] REGEV, O.: On lattices, learning with errors, random linear codes, and cryptography. STOC'05 : Proc. Thirty-Seventh Annual ACM Symposium on Theory of Computing, New York, NY, USA pp. 84–93 (2005). URL https://ci.nii.ac.jp/naid/20000814572/en/

[67] Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)

[68] The PARI Group, Bordeaux: PARI/GP https://pari.math.u-bordeaux.fr/download.html. URL https://pari.math.u-bordeaux.fr/download.html

[69] Wang, L., Aono, Y., Phong, L.T.: A New Secure Matrix Multiplication from Ring-LWE. In: S. Capkun, S.S.M. Chow (eds.) Cryptology and Network Security, pp. 93–111. Springer International Publishing, Cham (2018)

[70] Wang, L., Hayashi, T., Aono, Y., Phong, L.T.: A Generic yet Efficient Method for Secure Inner Product. In: Z. Yan, R. Molva, W. Mazurczyk, R. Kantola (eds.) Network and System Security, pp. 217–232. Springer International Publishing, Cham (2017)

[71] Wu, D., Haven, J.: Using Homomorphic Encryption for Large Scale Statistical Analysis (2012). URL https://crypto.stanford.edu/~dwu4/FHE-SI_Report.pdf

[72] Yann LeCun, C.C., Burges, C.J.: The MNIST Database of Handwritten Digits (1998). URL http://yann.lecun.com/exdb/mnist/

[73] Yasuda, M., Shimoyama, T., Kogure, J., Yokoyama, K., Koshiba, T.: Secure pattern matching using somewhat homomorphic encryption. In: Proceedings of the 2013 ACM workshop on Cloud computing security workshop–CCSW 2013, pp. 65–76. ACM (2013)

[74] Yasuda, M., Shimoyama, T., Kogure, J., Yokoyama, K., Koshiba, T.: New Packing Method in Somewhat Homomorphic Encryption and Its Applications. Security and Communication Networks **8**(13), 2194–2213 (2015)

[75] Yasuda, M., Shimoyama, T., Kogure, J., Yokoyama, K., Koshiba, T.: Secure Statistical Analysis Using RLWE-Based Homomorphic Encryption. In: Australasian Conference on Information Security and Privacy–ACISP 2015, *Lecture Notes in Computer Science*, vol. 9144, pp. 471–487. Springer (2015)

Academic Records

Journal

- Duong Hoang Dung, Pradeep Kumar Mishra, Masaya Yasuda: "Efficient secure matrix multiplications using RLWE-based homomorphic encryption". Tatra Mountains Mathematical Publications, Volume 67, Issue 1, Pages 69-83, ISSN.

Conference

- Deevashwer Rathee, Pradeep Kumar Mishra, and Masaya Yasuda. 2018. Faster PCA and Linear Regression through Hypercubes in HElib. In Proceedings of the 2018 Workshop on Privacy in the Electronic Society (WPES'18). ACM, New York, NY, USA, 42-53. DOI: https://doi.org/10.1145/3267323.3268952

- Quoc Huy Le, Pradeep Kumar Mishra, Duong Hoang Dung and Masaya Yasuda: (2018) Solving LWR via BDD Strategy: Modulus Switching Approach. In: Camenisch J., Papadimitratos P. (eds) Cryptology and Network Security. CANS 2018. Lecture Notes in Computer Science, vol 11124. Springer, Cham.

- Pradeep Kumar Mishra, Duong Hoang Dung, and Masaya Yasuda: "Enhancement for Secure Multiple Matrix Multiplications over Ring-LWE Homomorphic Encryption". In: Liu J., Samarati P. (eds) Information Security Practice and Experience. ISPEC 2017. Lecture Notes in Computer Science, vol 10701. Springer, Cham.

- Pradeep Kumar Mishra, Duong Hoang Dung, and Masaya Yasuda: “Efficient secure matrix multiplications using RLWE-based homomorphic encryption”. In Central European Conference on Cryptology (CECC-2016).

Preprint

- Pradeep Kumar Mishra, Deevashwer Rathee, Dung Hoang Duong, and Masaya Yasuda. Fast secure matrix multiplications over ring-based homomorphic encryption. Cryptology ePrint Archive, Report 2018/663, 2018. https://eprint.iacr.org/2018/663.

Talks and Posters

- 王立華，Pradeep Kumar Mishra，青野良範，Le Trieu Phong, 安田雅哉，Ring-LWEを用いたセキュアな行列乗算のためのパッキング方法，SCIS 2018.
- Pradeep Kumar Mishra, Duong Hoang Dung, and Masaya Yasuda: Secure Multiple Matrix Multiplications via Homomorphic Encryption , SCIS 2018.
- Pradeep Kumar Mishra, Duong Hoang Dung, and Masaya Yasuda: Efficient Packing Method For Secure Matrix Multiplication Over Ring-LWE Somewhat Homomorphic Encryption, SCIS 2017.
- Pradeep Kumar Mishra, Duong Hoang Dung, and Masaya Yasuda: Efficient Matrix Packing using Ring-LWE based Homomorphic Encryption, Poster in the Forum “Math-for-Industry” 2017 at the University of Hawaii, Honolulu, USA.
- Pradeep Kumar Mishra, Duong Hoang Dung, and Masaya Yasuda: Faster Matrix Multiplication using Ring-based Homomorphic Encryption, Poster in the Forum “Institute of Mathematics for Industry” 2017 at the Kyushu University, Fukuoka, Japan.