IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 4, 2020, accepted May 14, 2020, date of publication May 20, 2020, date of current version June 5, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2996000

seIMC: A GSW-Based Secure and Efficient Integer Matrix Computation Scheme With Implementation

YANAN BAI[1,2], XIAOYU SHI[3], (Member, IEEE), WENYUAN WU[1], JINGWEI CHEN[1], AND YONG FENG[1]

[1]Chongqing Key Laboratory Automated Reasoning and Cognition, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
[3]Chongqing Key Laboratory of Big Data and Intelligent Computing, Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China

Corresponding author: Yong Feng (yongfeng@cigit.ac.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 11671377, in part by the Chongqing Science and Technology Program under Grant cstc2018jcyj-yszxX0002 and Grant cstc2019yszx-jcyjX0003, in part by the Guizhou Science and Technology Program under Grant [2020]4Y056, in part by the Chongqing Research Program of the Key Standard Technologies Innovation of Key Industries under Grant cstc2017zdcy-zdyfX0076, in part by the Chongqing Research Program of Technology Innovation and Application under Grant 2019jscx-zdztzxX0019, and in part by the Chongqing Natural Science Foundation under Grant cstc2019jcyj-msxmX0638.

ABSTRACT As atomic operations, secure matrix-based computations using homomorphic encryption (HE) have attracted much attention in cloud-based machine learning. However, most existing secure matrix computation solutions that focus on HE schemes suffer efficiency loss as the size of the matrix, which greatly limits their applications in the big data environment. To address these issues, this paper proposes seIMC, an integer matrix computation scheme based on the Gentry-Sahai-Waters (GSW) scheme, to cope with privacy protection and secure computation of large-scale data. In detail, we translate the GSW scheme to encrypt an integer matrix modulo q (i.e., a large positive integer), and homomorphically compute matrix addition and multiplication, which is a natural extension of HAO scheme. Besides, the correctness and security analysis of seIMC are shown, and complexity analysis is also given in this study. Furthermore, the proposed schemes are implemented, including public-key encryption and private-key encryption schemes. Compared with existing secure matrix computation schemes, the proposed scheme performs better in execution time. Finally, seIMC is applied to solve the problem of the number of ways in which any two participants make friends through k steps in an encrypted social network. Experiments show that when the cloud server processes an integer matrix of 1000 people with a security level of 90, namely, 1 million data volumes, it only takes approximately 1.9 minutes for each homomorphic matrix multiplication. Hence, the practicality of the proposed seIMC in privacy protection under a big data environment is highly proven.

INDEX TERMS Homomorphic encryption, matrix computation, machine learning, GSW encryption scheme, big data, privacy protection.

I. INTRODUCTION

Fueled by the massive influx of data, extensive computational resources and advanced machine learning algorithms, artificial intelligence (AI) applications such as automatic driving, face recognition and smart homes have quickly entered people's lives [1]–[3]. Dependent on the powerful computation and storage abilities of cloud computing, a growing number of AI applications have migrated to the cloud to train their model with large-scale datasets by renting cloud-based machine learning services. In addition, increasing amounts of data and individual privacy information are collected and processed continuously in untrusted cloud servers [4]. Therefore, it poses a natural and important question on the

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

cloud-based machine learning that whether we can store and train such private data and model to the cloud environment in a secure manner, e.g., this issue is extremely relevant in social networks due to privacy concerns about individual sensitive information. This is also the case in the field of biomedicine concerning patients' private data.

In this context, homomorphic encryption (HE) is one of the most promising approaches for addressing this challenge and receives much attention in both academic and industry [5]–[7]. Compared to secure multiparty computation and differential privacy technologies, homomorphic encryption has obvious advantages in supporting noninteractive operations and saving on communication costs, allowing us to evaluate functions over encrypted messages directly to obtain the same results as evaluating the corresponding plaintext [8], [9]. Based on these properties, AI applications integrated with HE technologies can effectively defend against attacks on data and models in untrusted third-party environments (e.g., cloud environments).

Regarding HE, Rivest *et al.* [10] first proposed the concept of privacy homomorphism in 1978 and adopted the homomorphism of encryption functions to protect data privacy. Since then, cryptology researchers have carried out extensive studies on homomorphic encryption [11]. For example, Elgamal [12] proposed a homomorphic encryption scheme that supports the multiplication operation. Paillier [13] designed a public-key homomorphic encryption system based on composite degree residuality classes, which can implement the addition operation on the encrypted data. These above solutions are called partially homomorphic encryption schemes (PHEs), which only support particular operations (e.g., addition or multiplication) on encrypted data with a limited number of times.

Until 2009, Gentry proposed the first plausible and achievable fully homomorphic encryption scheme (FHE) [14], which is based on the idea of mathematical lattices. However, it is a conceptually and practically unrealistic scheme. To control the noise growth, FHE requires the refreshing of ciphertexts frequently via bootstrapping technology, which incurs a heavy extra computational cost. After that, some bounded (leveled) FHE schemes (LHE) were proposed to make FHE more practical. For example, Brakerski and Vaikuntanathan proposed an efficient fully homomorphic encryption from the standard learning with error (LWE) problem [15]. In 2014, Brakerski, Gentry and Vaikuntanathan proposed the BGV [16] that utilizes the efficiency feature of the ring-learning with error (RLWE) problem to build HE schemes. With the introduction of new techniques (e.g., bit decomposition, module switching, key switching), HELIB [17] is the implementation of the BGV scheme, which uses the high-performance mathematical function library NTL. These BGV-based solutions are deemed second-generation HE schemes. However, these second-generation HE solutions also bring extra computation costs due to unnatural key switching. To address this issue, Gentry, Sahai and Waters proposed GSW [18] that encrypts the plaintexts using the approximate eigenvalues of the ciphertext matrix with the eigenvector as the secret key. GSW can largely reduce the unnecessary key switching brought by BGV-based solutions since the ciphertext computation of GSW is based on the matrix computation directly, which does not need to obtain the user's evaluation key. GSW-based solutions are called the third-generation of HE schemes. Note that as the weaker version of FHE, LHE can only support depth-bounded homomorphic operations (i.e., addition and multiplication) without bootstrapping, but it is applicable in various scenarios with reasonable performances. Hence, we focus on building our solution on the LHE scheme [8]. Furthermore, in this paper, we focus on constructing a secure and efficient integer matrix computation solution from the third-generation of HE schemes (i.e., GSW-based).

Currently, matrix-based computations are the core and atomic operations for major AI applications. 1). The collected data are often organized in the form of a matrix, which can be found in many domains, e.g., social network services dealing with friend relationships [19], the protein interaction network in bioinformatics [20], business intelligence from user-item rating data in recommendation systems [21], [34]. 2). Various tasks are executed via matrix-based computations, such as massive data-based statistical analysis, the training process of deep learning models, and the prediction task of learned neural network models. 3). In the cloud environment, the data are rarely organized in a bit matrix or an integer vector, in most cases they are organized in a rational number matrix. At the same time, in the allowable precision range, the fraction can be converted into an integer for approximate computation. Therefore, secure integer matrix computation becomes an important issue in the cloud. As a result, it is of great practical significance to establish an efficient and feasible integer matrix homomorphic encryption scheme for cloud-based machine learning.

Following this line, some secure matrix computation schemes based on HE have been proposed. For example, Wu and Haven *et al.* proposed a safety inner product method on packed ciphertexts using the single-instruction-multi-data (SIMD) approach [22]. It encrypts the rows or columns of the matrix as vectors and computes the result using homomorphism multiplication as the inner product of the two encrypted vectors. Duong *et al.* [23] packed the target matrix into a single ciphertext in polynomial form and then performed a homomorphic multiplication on the packed ciphertext over RLWE. Based on this method, Mishra *et al.* [24] built an enhanced version of the secure matrix multiplication proposed by Duong, but there are useless terms in the ciphertexts. As a result, these useless terms in the plaintext polynomials are eliminated by encrypting and recoding the plaintext information; thus it is only suitable for a one-depth homomorphic multiplication scenario, since it leads to a large expansion rate of ciphertexts.To address it, Jiang *et al.* [25] presented a novel matrix encoding method that can encrypt more than one matrix in a single ciphertext and adapted an efficient evaluation strategy for generic matrix operations via linear

transformations. However, this method suffers efficiency loss when dealing with large-scale data. Taking advantage of the GSW scheme in which homomorphic addition and multiplication are just natural matrix addition and multiplication, Hiromasa *et al.* [27] first conducted a GSW-FHE scheme for matrix homomorphism computations (i.e., HAO) and optimized the bootstrapping technique proposed by [26]. However, all these improvements target binary plaintext, which greatly restricts its application in the real world.

To address the above issues, this paper proposes seIMC, a novel GSW-based integer matrix computation scheme. Different from the original GSW scheme that encrypts a bit or an integer into a ciphertext matrix, the seIMC directly encrypts a whole plaintext matrix into a ciphertext matrix to reduce the homomorphic computing time. The proposed scheme extends the plaintext space of the HAO scheme, which can encrypt not only the bit matrix but also the integer matrix modulo q, where q is a large positive integer. In addition, it supports the homomorphic computation of matrix addition and multiplication. The main contributions of this paper are as follows:

- We propose a secure and efficient GSW-based integer matrix computation scheme for public-key encryption and private-key encryption, where the matrix element is an integer with modulo q. In detail, it includes the encryption algorithm, decryption algorithm, and generic homomorphic operations such as addition and multiplication.
- We give the security proofs and correctness analysis of the proposed scheme. Computational complexity analysis of the encryption, decryption and homomorphic algorithms are also given.
- We conduct extensive experiments to evaluate the efficiency of the proposed seIMC scheme in terms of encryption, decryption and homomorphic operations. Most importantly, we apply seIMC to solve the graph theory problem in social networks.

The rest of this paper is organized as follows. Section II gives the preliminaries. Section III describes the proposed seIMC scheme in detail, and theoretical analyses are also given. Section IV empirically evaluates seIMC. Section V shows the application to social networks. Finally, Section VI discusses and concludes this paper.

II. PRELIMINARIES

A. NOTATIONS

We use the symbol $\mathbb{Q}$ to denote the set of rational numbers, and $\mathbb{R}$ is the set of real numbers, while $\mathbb{N}$ and $\mathbb{Z}$ are the sets of natural numbers and integers, respectively. Let $\boldsymbol{D}$ be some group, and $\boldsymbol{P}$ be some probability distribution, then we use $a \xleftarrow{\mathrm{U}} \boldsymbol{D}$ to denote that a is chosen from $\boldsymbol{D}$ uniformly at random, and use $b \xleftarrow{R} \boldsymbol{P}$ to denote that b is chosen along with $\boldsymbol{P}$.

Assume that vectors are in column form and are written using bold lower-case letters, e.g., $\boldsymbol{x}$, where x_i represents the ith element of a vector $\boldsymbol{x}$. We denote the ℓ_∞ norm (i.e., the maximum norm) of the vector $\boldsymbol{x}$ by $\|\boldsymbol{x}\|_\infty$. The inner product between two vectors is defined as $<\boldsymbol{x}, \boldsymbol{y}>= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$. Similarly, we use bold capital letters to denote matrices, e.g., $\boldsymbol{X}$. For a matrix $\boldsymbol{X} \in \mathbb{Z}^{m\times n}$, $\|\boldsymbol{X}\|_\infty := \max_{i\in[n]}\{\|\boldsymbol{x}_i\|_\infty\}$ denotes the ℓ_∞ of $\boldsymbol{X}$, where $\boldsymbol{x}_i$ denotes the ith column vector of $\boldsymbol{X}$. Meanwhile, $\boldsymbol{X}^T \in \mathbb{Z}^{n\times m}$ denotes the transpose of $\boldsymbol{X}$. For matrices $\boldsymbol{X} \in \mathbb{Z}^{m\times n_1}$, $\boldsymbol{Y} \in \mathbb{Z}^{m\times n_2}$, $[\boldsymbol{X}|\boldsymbol{Y}] \in \mathbb{Z}^{m\times(n_1+n_2)}$ represents the column concatenation of $\boldsymbol{X}$ with $\boldsymbol{Y}$, while the row concatenation of $\boldsymbol{X} \in \mathbb{Z}^{m_1\times n}$ with $\boldsymbol{Y} \in \mathbb{Z}^{m_2\times n}$ is $\left[\frac{\boldsymbol{Y}}{\boldsymbol{X}}\right] \in \mathbb{Z}^{(m_1+m_2)\times n}$.

B. THE LEARNING WITH ERRORS (LWE)

LWE is considered as one of the hardest problems to solve in regular time, even when using advanced quantum computing technology. It was first introduced by Regev [29]. The definition of the decisional version LWE is:

Definition 1 (DLWE): Given a security parameter λ, let $n := n(\lambda)$ be an integer dimension, $q := q(\lambda) \geq 2$ be an integer modulus, and $\chi := \chi(\lambda)$ be an error distribution over $\mathbb{Z}$. Then, the $\mathbf{DLWE}_{n,q,\chi}$ *is the problem to distinguish the following two distributions:*

- *The first distribution: sample a tuple $(\boldsymbol{a}_i, b_i)$ from uniformly over $\mathbb{Z}_q^n \times \mathbb{Z}_q$;*
- *The second distribution: when $\boldsymbol{s} \xleftarrow{U} \mathbb{Z}_q^n$, then a tuple $(\boldsymbol{a}_i, b_i)$ is chosen by sampling $\boldsymbol{s} \xleftarrow{U} \mathbb{Z}_q^n$, $e_i \xleftarrow{R} \chi$, and setting $b_i := <\boldsymbol{a}_i, \boldsymbol{s}>+e_i \bmod q$.*

The $\mathbf{DLWE}_{n,q,\chi}$ *assumption is that the* $\mathbf{DLWE}_{n,q,\chi}$ *is infeasible.*

Regev reduced the hardness of worst-case lattice problems such as $\mathbf{GapSVP}_\gamma$ and $\mathbf{SIVP}_\gamma$ to the $\mathbf{DLWE}_{n,q,\chi}$ problem. In detail, $\mathbf{GapSVP}_\gamma$ has the problem of distinguishing between the case in which the lattice has a vector shorter than $r \in \mathbb{Q}$ and the case in which all the lattice vectors are greater than $\gamma \cdot r$. The hardness of $\mathbf{SIVP}_\gamma$ is defined to find the set of short linearly independent vectors in a lattice. The reductions are described in *Corollary 1*.

Corollary 1 [29]–[31]: Let $q = q(n) \in \mathbb{N}$ be a power of primes $q := p^r$ or a product of distinct prime numbers $q := \Pi_i q_i$ such that $q_i :=$poly(n) for all i, let $\alpha \geq \sqrt{n}/q$. If there is an efficient algorithm that solves the (average-case) $\mathbf{DLWE}_{n,q,\chi}$ *problem, then:*

- *There is an efficient quantum algorithm that solves* $\mathbf{GapSVP}_{\tilde{O}(n/\alpha)}$ *and* $\mathbf{SIVP}_{\tilde{O}(n/\alpha)}$ *in the worst-case for any n-dimensional lattices.*
- *If an addition $q \geq \tilde{O}\left(2^{n/2}\right)$, there exists an efficient classical algorithm to solve* $\mathbf{GapSVP}_{\tilde{O}(n/\alpha)}$ *in the worst-case for any n-dimensional lattices.*

C. HOMOMORPHIC ENCRYPTION, CIRCULAR SECURITY

An HE scheme consists of four algorithms, HE = (**Keygen**, **Enc**, **Dec**, **Eval**), and is illustrated as follows:

- **KeyGen**(1^λ): it returns a secret key $\boldsymbol{sk}$, a public key $\boldsymbol{pk}$ and a public evaluation key $\boldsymbol{evk}$.

- $\mathbf{Enc_{pk}}(m)$: it encrypts a plaintext $m \in \mathcal{M}$ into a ciphertext $c \in \mathcal{C}$ by using public key $\boldsymbol{pk}$.
- $\mathbf{Dec_{sk}}(c)$: it recovers the original plaintext m from the ciphertext c via the secret key $\boldsymbol{sk}$.
- $\mathbf{Eval_{evk}}(f, c_1, \ldots, c_k)$: using the evaluation key $\boldsymbol{evk}$, the ciphertext $c \in \mathcal{C}$ can be computed by using the function $f: \mathcal{M}^k \to \mathcal{M}$ to $c_1, \ldots, c_k$.

To prove the security of an HE scheme, we introduce a special kind of circular security as follows.

Definition 2 (Circular Security): Denote κ as the keyspace defined by the security parameter λ, and $\mathcal{M}$ and $\mathcal{C}$ are the plaintext and ciphertext space, respectively. f is a function from $\mathcal{M}$ to $\mathcal{C}$. For all probabilistic polynomial-time adversaries $\mathcal{A}$, the homomorphic encryption scheme **HE = (Keygen, Enc, Dec, Eval) is circular security with respect to** *f when the advantage of $\mathcal{A}$ can be negligible in the following games:*

- A challenger calculates $(\boldsymbol{pk}, \boldsymbol{sk}, \boldsymbol{evk}) \leftarrow \mathbf{KeyGen}(1^\lambda)$ and selects a bit $b \leftarrow \{0, 1\}$;
- Define the function f_+ as $\mathcal{M} \times \mathcal{M} \to \mathcal{M}$, and $f_+(x, y) := x + y \in \mathcal{M}$; then, the challenger computes a ciphertext c^* as follows and sends c^* to $\mathcal{A}$.

$$\boldsymbol{c}^* := \begin{cases} \mathbf{Eval_{evk}}(f_+, \mathbf{Enc_{pk}}(0), f(\boldsymbol{sk})) & \text{if } b = 0 \\ \mathbf{Enc_{pk}}(0) \in \boldsymbol{C} & \text{otherwise} \end{cases} \quad (1)$$

- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$

The advantage of $\mathcal{A}$ is $\Pr[b = b'] - 1/2$.

D. GSW SCHEME

The original GSW scheme was proposed by Gentry, Sahai, Waters [19]. It adopts the approximate eigenvector method based on the plaintext space $\mathcal{M}$ to construct the ciphertext space $\mathcal{C}$. We first introduce the Gadget matrix $\boldsymbol{G}$ and the randomized function $\mathbf{G}^{-1}$.

Lemma 1 [32]: Let matrix $\boldsymbol{C} \in \mathbb{Z}_q^{n \times m}$, there is a fixed and primitive matrix $\boldsymbol{G} \in \mathbb{Z}_q^{n \times n\ell}$ and a deterministic, randomized function $\boldsymbol{G}^{-1} : \mathbb{Z}_q^{n \times m} \to \mathbb{Z}_q^{n\ell \times m}$ such that $\boldsymbol{X} \xleftarrow{\mathrm{R}} \mathbf{G}^{-1}(\boldsymbol{C})$ is sub-gaussian with parameter O(1) and always satisfies $\boldsymbol{GX}=\boldsymbol{C}$.

Let $\ell = \lceil \log_2 q \rceil$, $\boldsymbol{g}^{\mathrm{T}} = (2^0, 2^1, \ldots 2^{\ell-1})$, and $\boldsymbol{I_n}$ be the $n \times n$ identity matrix, we define the Gadget matrix as $\boldsymbol{G} := \boldsymbol{I}_n \otimes \boldsymbol{g}^{\mathrm{T}} \in \mathbb{Z}_q^{n \times n\ell}$, where $\otimes$ is the tensor multiplication operation. The definition of sub-gaussian distribution can be found in [26].

For example, let $\ell = 3$, $n = 2$, $q = 8$, $\boldsymbol{G}=\boldsymbol{I}_2 \otimes (2^0, 2^1, 2^2) = \begin{bmatrix} 1 & 2 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 4 \end{bmatrix} \in \mathbb{Z}_8^{2 \times 6}$. If $\boldsymbol{C}=[6\ 7]^{\mathrm{T}} \in \mathbb{Z}_8^{2 \times 1}$, then $\mathbf{G}^{-1}(\boldsymbol{C})=[0\,1\,1\,1\,1\,1]^{\mathrm{T}} \in \{0, 1\}^{6 \times 1}, \boldsymbol{G}\mathbf{G}^{-1}(\boldsymbol{C}) = [6\ 7]^{\mathrm{T}} = \boldsymbol{C}$.

Next, we introduce the GSW scheme variant from [26], which is identical to the original GSW scheme except for the introduction of the Gadget matrix $\boldsymbol{G}$ and the randomized function $\mathbf{G}^{-1}$, as well as some syntactic differences. The encryption algorithm can be further divided into the public-key encryption algorithm [33] and the private-key encryption algorithm [26]. The private-key encryption scheme can be described as follows:

- **Setup**(λ, L): Given the security parameter λ and the circuit multiplication depth L, let $n := n(\lambda)$ be an integer dimension, $q := q(\lambda) \geq 2$ be an integer modulus, and $\chi := \chi(\lambda)$ be a sub-gaussian error distribution over $\mathbb{Z}$. Output $\boldsymbol{params} = (n, q, \chi, \ell, \boldsymbol{G})$.
- **KeyGen**$(\boldsymbol{params})$: Samples $\bar{\boldsymbol{s}} \xleftarrow{\mathrm{R}} \chi^{n-1}$, output the secret key $\boldsymbol{s} = [\bar{\boldsymbol{s}} \,|\, 1] \in \mathbb{Z}^n$.
- **Enc**$(\boldsymbol{params}, \bar{\boldsymbol{s}}, \mu \in \mathbb{Z}_q)$: $\bar{\boldsymbol{C}} \xleftarrow{\mathrm{U}} \mathbb{Z}_q^{(n-1) \times n\ell}$, $\boldsymbol{e} \xleftarrow{\mathrm{R}} \chi^{n\ell}$. Let $\boldsymbol{b}^T = \left[\boldsymbol{e}^T - \bar{\boldsymbol{s}}^T \bar{\boldsymbol{C}}\right]_q$, output the ciphertext

$$\boldsymbol{C} = \begin{pmatrix} \bar{\boldsymbol{C}} \\ \boldsymbol{b}^T \end{pmatrix} + \mu \boldsymbol{G} \quad (2)$$

- **Dec**$(\boldsymbol{params}, \boldsymbol{s}, \boldsymbol{C})$: For $q = 2^\ell$, select the last ℓ columns of $\boldsymbol{C}$ as $\mathbf{C}_{(\ell)}$. Then, $\boldsymbol{s}^T \boldsymbol{C}_{(\ell)} = \mu \cdot \boldsymbol{g}^T + \boldsymbol{e}'$, where $\boldsymbol{e}'$ is the error vector. Recover the Least Significant Bit $\mathbf{LSB}(\mu)$ from $(\mu - LSB(\mu)) \cdot 2^{\ell-2} + e'_{\ell-2}$, where e'_i is the ith element of $\boldsymbol{e}'$.

The correctness of the GSW scheme can be guaranteed by *Lemma 2*.

Lemma 2 [18], [33]: For security key $\boldsymbol{s}$, plaintext $\mu \in \mathbb{Z}_q$, and the ciphertext $\boldsymbol{C} \in \mathbb{Z}_q^{n \times n\ell}$, let the noise term be $\boldsymbol{e}^T$ such that $\boldsymbol{s}^T \boldsymbol{C} - \mu \boldsymbol{s}^T \boldsymbol{G} = \boldsymbol{e}^T$ mod q. If $\left\|\boldsymbol{e}^T\right\|_\infty < q/8$, then $\mathbf{Dec}(\boldsymbol{params}, \boldsymbol{s}, \boldsymbol{C})$ *can decrypt μ correctly.*

Since a fresh ciphertext is just $\mu \boldsymbol{G}$ plus a matrix of $n\ell$ independent LWE samples under secret $\bar{\boldsymbol{s}}$, the IND-CPA security of the above scheme follows from the assumed hardness of the $\mathbf{DLWE}_{n-1,q,\chi}$, where $\boldsymbol{C}$ is pseudorandom by assumption and hence hides $\mu \boldsymbol{G}$.

Note that the process of the public-key encryption scheme is similar to that of the private-key scheme, and the security of the public-key scheme follows directly from *Lemma 1* in [18].

III. PROPOSED seIMC SCHEME

In this section, we first propose a secure and efficient matrix computation scheme via homomorphic encryption, and then present the security proofs and complexity analysis of the proposed scheme. Finally, the correctness proof and efficiency analysis of the proposed scheme are also given.

As the third-generation of HE schemes, GSW realizes the encryption of plaintext as bits and integers modulo q. Hence, it is natural to pack plaintexts as vectors or matrices for encryption. In [27], it was demonstrated that a simple extension of plaintext space from bits to binary vectors cannot yield plaintext-slot-wise homomorphic operations (e.g., addition and multiplication). However, homomorphic plaintext-slot-wise operations can be supported by constructing matrices to store binary vectors in their diagonal entries. Based on this storage they constructed a matrix-based homomorphic encryption scheme HAO, which supports homomorphic binary matrix addition and multiplication. To extend the plaintext space of the HAO scheme to meet the requirements of large-scale AI applications in the real world, we

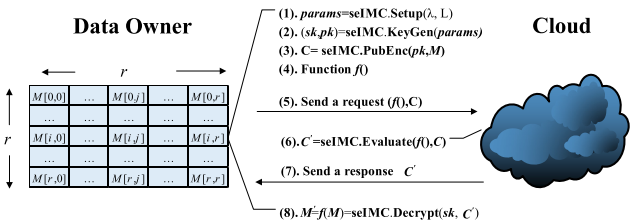


FIGURE 1. The overview of the seIMC public-key scheme.

construct the homomorphic integer matrix encryption scheme that encrypts integer modulo q, where q is a large positive integer.

The proposed seIMC public-key scheme is described in Fig. 1. It gives an example of a secure cloud computing organization. There are two sides: the data owner and the cloud server. By calling the functions seIMC.**Setup**() and seIMC.**KeyGen**(), the data owner first generates a key pair (i.e., the secret-key $\boldsymbol{sk}$ and the public key $\boldsymbol{pk}$) with a specified cryptographic security parameter λ and the multiplication depth of circuit L. Then, the data owner uses the $\boldsymbol{pk}$ to encrypt plaintext data $\boldsymbol{M}$ with size of $(r+1)\times(r+1)$ by calling seIMC. **PubEnc**(), where only the data owner knows the $\boldsymbol{sk}$. After that, the data owner sends the encrypted data to the cloud server, where the data are always kept in encrypted form $\boldsymbol{C}$. In addition, the data owner also sends the function $\boldsymbol{f}()$ to the cloud server if he needs some data that satisfy certain conditions. Taking advantage of the GSW-based scheme with the approximate eigenvector method, the data owner does not need to create and send the evaluation key ($\boldsymbol{evk}$) to the cloud server to support homomorphic operations. The cloud server receives the encrypted data $\boldsymbol{C}$ without $\boldsymbol{sk}$ and function $\boldsymbol{f}()$ from the data owner, and produces the necessary computations $\boldsymbol{f}(\boldsymbol{C})$ by calling seIMC. $\mathbf{Eval}_{\mathbf{add}}()$ or seIMC. $\mathbf{Eval}_{\mathbf{mult}}$ Then, result $\boldsymbol{C}'$ is sent to the data owner as a response. When the data owner receives the result $\boldsymbol{C}'$, he can decrypt the $\boldsymbol{C}'$ that satisfies the function $\boldsymbol{f}()$ by calling seIMC. **Decrypt**() with**DecOneNum**() and using the $\boldsymbol{sk}$.

A. CONSTRUCTION OF THE seIMC SCHEME

In seIMC, for an integer modulus q, we let $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ denote the quotient ring of integers modulo q. $\boldsymbol{s}$ is the column vector of the secret key matrix $\boldsymbol{S}$ and $\boldsymbol{e}$ is the column vector of the noise matrix $\boldsymbol{E}$. $\boldsymbol{A}$ and $\boldsymbol{R}$ are uniformly random matrices. $\boldsymbol{B}$ and $\boldsymbol{P}$ denote the public key matrices. Let $\boldsymbol{M} \in \mathbb{Z}_q^{r\times r}$ be the plaintext matrix with size $r\times r$ in $\mathbb{Z}_q$.

1) PARAMETERS SETUP ALGORITHM: SETUP()

Given the security parameter λ and the multiplication depth of circuit L, we let $\ell = \lceil \log_2 q \rceil$ and the integer modulus $q = q(\lambda, \mathrm{L}) := 2^{\ell}$. The lattice dimension $n = n(\lambda, L)$ and the noise distribution $\chi = \chi(\lambda, L)$ over $\mathbb{Z}$ that are assumed to be sub-gaussian. Then, let $m = m(\lambda, L) := \mathrm{O}((n+r)\ell)$, $N := (n+r)\ell$, and $\boldsymbol{G} = \boldsymbol{g}^T \otimes \boldsymbol{I}_{n+\mathrm{r}} \in \mathbb{Z}_q^{(n+r)\times N}$, where $\boldsymbol{g}^{\mathrm{T}} = \{2^0, 2^1, \ldots, 2^{\ell-1}\}$. The output of **Setup**(λ, L) is $\boldsymbol{params}$=(n, q, χ, m).

2) KEY GENERATION ALGORITHM: KEYGEN()

We sample matrix $\bar{\boldsymbol{S}} \xleftarrow{\mathrm{R}} \chi^{r\times n}$, and denote $\boldsymbol{I}_r$ as the $r\times r$ identity matrix, then the secret key matrix $\boldsymbol{sk}$ is computed by:

$$\boldsymbol{S} = [\boldsymbol{I}_{\boldsymbol{r}} \parallel -\bar{\boldsymbol{S}}] \in \mathbb{Z}_q^{r\times(n+r)} \tag{3}$$

For the public key matrix $\boldsymbol{pk}$, we sample a uniformly random matrix $\boldsymbol{A} \xleftarrow{\mathrm{U}} \mathbb{Z}_q^{n\times m}$ and a random noise matrix $\boldsymbol{E} \xleftarrow{\mathrm{R}} \chi^{r\times m}$; then, the public key matrix can be computed by:

$$\boldsymbol{B} := \begin{pmatrix} \bar{\boldsymbol{S}}\boldsymbol{A} + \boldsymbol{E} \\ \boldsymbol{A} \end{pmatrix} \in \mathbb{Z}_q^{(n+r)\times m} \tag{4}$$

Let $\boldsymbol{M}_{(i,j)} \in \{0,1\}^{r\times r}$ $(i,j = 1,\ldots,r)$ denote the matrix with 1 in the ith row and jth column and 0 in the others. For all $i,j = \{1,\ldots,r\}$, we sample $\boldsymbol{R}_{(i,j)} \xleftarrow{\mathrm{U}} \{0,1\}^{m\times N}$ uniformly, then compute

$$\boldsymbol{P}_{(i,j)} := \boldsymbol{B}\boldsymbol{R}_{(i,j)} + \begin{pmatrix} \boldsymbol{M}_{(\boldsymbol{i},\boldsymbol{j})}\boldsymbol{S} \\ 0 \end{pmatrix} \boldsymbol{G} \in \mathbb{Z}_q^{(n+r)\times N} \tag{5}$$

Finally, the output of **KeyGen**($\boldsymbol{params}$) is $\boldsymbol{sk}$:=$\boldsymbol{S}$, $\boldsymbol{pk}$:= $\{(\boldsymbol{P}_{(i,j)}, \boldsymbol{B}) \| 1 \le i,j \le r\}$.

3) ENCRYPTION ALGORITHMS: SECENC() AND PUBENC()

To meet the requirements of the encryption algorithm in different scenarios, we conduct the public-key encryption algorithm (i.e., $\mathbf{PubEnc}_{\mathbf{pk}}(\boldsymbol{M})$) and private-key encryption algorithm (i.e., $\mathbf{SecEnc}_{\mathbf{sk}}(\boldsymbol{M})$).

- $\mathbf{SecEnc}_{\mathbf{sk}}(\boldsymbol{M})$: Sample the random matrix $\bar{\boldsymbol{A}} \xleftarrow{\mathrm{U}} \mathbb{Z}_q^{n\times N}$ and $\boldsymbol{E} \xleftarrow{\mathrm{R}} \chi^{r\times N}$; then, the ciphertext $\boldsymbol{C}$ can be computed by:

$$\boldsymbol{C} := \begin{pmatrix} \overline{\boldsymbol{S}\boldsymbol{A}} + \boldsymbol{E} \\ \bar{\boldsymbol{A}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{M}\boldsymbol{S} \\ 0 \end{pmatrix} \boldsymbol{G} \in \mathbb{Z}_q^{(n+r)\times N} \tag{6}$$

- $\mathbf{PubEnc}_{\mathbf{pk}}(\boldsymbol{M})$: Sample a random matrix $\boldsymbol{R} \xleftarrow{\mathrm{U}} \{0,1\}^{m\times N}$; then, the ciphertext $\boldsymbol{C}$ can be computed by:

$$\boldsymbol{C} := \boldsymbol{B}\boldsymbol{R} + \sum_{i=1}^{r}\sum_{j=1}^{r} M[i,j]\cdot \boldsymbol{P}_{(i,j)} \in \mathbb{Z}_q^{(n+r)\times N} \tag{7}$$

where $M[i,j]$ denotes the element of $\boldsymbol{M}$ in the ith row and jth column.

4) DECRYPTION ALGORITHMS: DECRYPT()

Note that for the public-key and private-key encryption algorithms, the decryption algorithms are the same. Let $\boldsymbol{C}$ be the input of **Dec**(), then the output is the plaintext $\boldsymbol{M}$. In detail, the decryption processing can be described as follows:

Step 1: compute the matrix $\boldsymbol{H}$ as

$$\boldsymbol{H} = \boldsymbol{S}\boldsymbol{C} = \boldsymbol{M}\boldsymbol{S}\boldsymbol{G} + \boldsymbol{E} \in \mathbb{Z}_q^{r\times N} \tag{8}$$

according to (6) or (7).

Algorithm 1 Pseudocode of function **DecOneNum**()

Input: ***params*** and vector $\gamma = (\gamma_0, \ldots, \gamma_{\ell-2})$
Output: integer μ
1: $x_0 = \text{round}(\gamma_{\ell-2}/2^{\ell-2}) \bmod 2$;
2: $\mu = x_0$, $\eta = 0$, $\zeta = 0$;
3: **For** $(i = 1; i <= \ell - 2; i++)$
4: $\quad \zeta = \eta/2 + 2^{\ell-3}x_0 \bmod q$;
5: $\quad x' = \text{round}((\gamma_{\ell-2-i} - \zeta)/2^{\ell-2}) \bmod 2$;
6: $\quad \mu = \mu + 2^i x \bmod q$;
7: $\quad \eta = \zeta$;
8: $\quad x_0 = x'$;
9: **End For**
10: **return** μ

Step 2: pack the first $r\ell$ columns of $\boldsymbol{H}$ as $\boldsymbol{H'} \in \mathbb{Z}_q^{r\times r\ell}$ and denote the noise matrix $\boldsymbol{E'} \in \mathbb{Z}_q^{r\times N}$ as the first $r\ell$ columns of $\boldsymbol{E}$. Then, we can obtain that $\boldsymbol{H'} = \boldsymbol{E'}+$, as shown at the bottom of this page.

Step 3: recover each element in the plaintext matrix $\boldsymbol{M}$ (i.e., $M[i,j] = \sum_{k=0}^{\ell-2} 2^k \cdot x_k$, where x_k denotes the k-bit of x via the function **DecOneNum**(γ, ***params***). In detail, the function **DecOneNum**($\boldsymbol{H'}[i, j\ell : (j+1)\,\ell - 2]$, ***params***) works as described in *Algorithm 1*, where vector $\boldsymbol{H'}[i, j\ell : (j+1)\,\ell - 2]$ represents the values in ith row of $\boldsymbol{H'}$ with columns from $j\ell$th to $((j+1)\ell$-2)th. For example, set $i = 0$ and $j = 0$, i.e., the input of **DecOneNum**() is $\boldsymbol{H'}$[0, 0:(l-2)], and then the output will be M[0, 0]. Based on this, we can recover all elements of $\boldsymbol{M}$ by iteratively using **DecOneNum**

5) HOMOMORPHIC MATRIX OPERATIONS: EVAL()

For the ciphertext matrices $\boldsymbol{C_1} \in \mathbb{Z}_q^{(n+r)\times N}$ and $\boldsymbol{C_2} \in \mathbb{Z}_q^{(n+r)\times N}$, homomorphic addition (i.e., $\mathbf{Eval_{Add}}$ $(\boldsymbol{C_1}, \boldsymbol{C_2})$) is defined as:

$$\mathbf{C_{add}} := \boldsymbol{C_1} + \boldsymbol{C_2} \in \mathbb{Z}_q^{(n+r)\times N} \tag{9}$$

For homomorphic multiplication (i.e., $\mathbf{Eval_{Mult}}(\boldsymbol{C_1}, \boldsymbol{C_2})$), $\boldsymbol{G^{-1}}(\boldsymbol{C_2}) \in \{0,1\}^{N\times N}$ is computed first, and then outputs:

$$\boldsymbol{C_1 G^{-1}(C_2)} \in \mathbb{Z}_q^{(n+r)\times N} \tag{10}$$

B. SCHEME ANALYSIS

1) CORRECTNESS ANALYSIS

The correctness of seIMC in the form of a public-key scheme and private-key scheme can be guaranteed by *Lemma 3* and *Lemma 4*, respectively.

Lemma 3: In the case of the public-key scheme, if the plaintext $\boldsymbol{M} \in \mathbb{Z}_q^{r\times r}$ can be encrypted to ciphertext $\boldsymbol{C}$ with the noise matrix $\boldsymbol{E} \xleftarrow{R} \chi^{r\times m}$ such that $(1+\rho)\,\|\boldsymbol{E}\|_\infty < q/8$ and $\sum_{j=0}^{r-1}\sum_{i=0}^{r-1} |M[i,j]| = \rho$, then the output of $\boldsymbol{Dec_{sk}(C)}$ is $\boldsymbol{M}$.

Proof: According to (5) and (7), we know that $\boldsymbol{SB} = [\boldsymbol{I}_r \| -\overline{\boldsymbol{S}}]\begin{bmatrix}\frac{\overline{\boldsymbol{S}}\boldsymbol{A}+\boldsymbol{E}}{\boldsymbol{A}}\end{bmatrix} = \boldsymbol{E}$, $\sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j]\cdot\boldsymbol{M}_{(i,j)} = \boldsymbol{M}$ then,

$$\begin{aligned}
\boldsymbol{H} = \boldsymbol{SC} &= \boldsymbol{S}\left(\boldsymbol{BR} + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j]\boldsymbol{P}_{(i,j)}\right) \\
&= \boldsymbol{S}\left(\boldsymbol{BR} + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j]\left(\boldsymbol{BR}_{(i,j)} + \begin{bmatrix}\boldsymbol{M}_{(i,j)}\boldsymbol{S} \\ \boldsymbol{0}\end{bmatrix}\boldsymbol{G}\right)\right) \\
&= \boldsymbol{ER} + \boldsymbol{ER}_{(i,j)} + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j]\boldsymbol{S}\begin{bmatrix}\boldsymbol{M}_{(i,j)}\boldsymbol{S} \\ \boldsymbol{0}\end{bmatrix}\boldsymbol{G} \\
&= \boldsymbol{ER} + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j]\boldsymbol{ER}_{(i,j)} \\
&\quad + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j][\boldsymbol{I}_r \| -\overline{\boldsymbol{S}}]\begin{bmatrix}\boldsymbol{M}_{(i,j)}\boldsymbol{S} \\ \boldsymbol{0}\end{bmatrix}\boldsymbol{G} \\
&= \boldsymbol{E}\left(\boldsymbol{R} + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j]\boldsymbol{R}_{(i,j)}\right) + \boldsymbol{MSG}
\end{aligned} \tag{11}$$

where $\boldsymbol{E}\left(\boldsymbol{R} + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j]\boldsymbol{R}_{(i,j)}\right) \in \mathbb{Z}_q^{r\times N}$ is the noise term. If $\left\|\boldsymbol{E}\left(\boldsymbol{R} + \sum_{i=0}^{r-1}\sum_{j=0}^{r-1} \boldsymbol{M}[i,j]\boldsymbol{R}_{(i,j)}\right)\right\|_\infty < q/8$ for all i, $j \in \{0, \ldots, r-1\}$, then it can be decrypted correctly when the following is satisfied: $\sum_{j=0}^{r-1}\sum_{i=0}^{r-1} |M[i,j]| = \rho$, then $(1+\rho)\,\|\boldsymbol{E}\|_\infty < q/8$, according to *Lemma 2*.

Lemma 4: In the case of the private-key scheme, if a plaintext $\boldsymbol{M} \in \mathbb{Z}_q^{r\times r}$ is encrypted to a ciphertext $\boldsymbol{C}$ with noise matrix $\boldsymbol{E}$ such that $\|\boldsymbol{E}\|_\infty < q/8$, then $\boldsymbol{Dec_{sk}(C)} = \boldsymbol{M}$.

Proof: We have:

$$\begin{aligned}
\boldsymbol{SC} &= \boldsymbol{S}\left(\begin{bmatrix}\frac{\overline{\boldsymbol{S}}\boldsymbol{A}+\boldsymbol{E}}{\boldsymbol{A}}\end{bmatrix} + \begin{bmatrix}\boldsymbol{MS} \\ \boldsymbol{0}\end{bmatrix}\right)\boldsymbol{G} \\
&= [\boldsymbol{I}_r \| -\overline{\boldsymbol{S}}]\begin{bmatrix}\frac{\overline{\boldsymbol{S}}\boldsymbol{A}+\boldsymbol{E}}{\boldsymbol{A}}\end{bmatrix} + [\boldsymbol{I}_r \| -\overline{\boldsymbol{S}}]\begin{bmatrix}\boldsymbol{MS} \\ \boldsymbol{0}\end{bmatrix}\boldsymbol{G} \\
&= \boldsymbol{E} + \boldsymbol{MSG}
\end{aligned} \tag{12}$$

According to *Lemma 2*, we know that if $\|\boldsymbol{E}\|_\infty < q/8$, then the private-key scheme can be decrypted correctly.

$$\left[\begin{array}{cccc|c|cccc}
M[0,0] & 2M[0,0] & \cdots & 2^{\ell-1}M[0,0] & \cdots\cdots & M[0,r-1] & 2M[0,r-1] & \cdots & 2^{\ell-1}M[0,r-1] \\
\vdots & \vdots & \vdots & \vdots & \vdots\ \vdots & \vdots & \vdots & \vdots & \vdots \\
M[r-1,0] & 2M[r-1,0] & \cdots & 2^{\ell-1}M[r-1,0] & \cdots\cdots & M[r-1,r-1] & 2M[r-1,r-1] & \cdots & 2^{\ell-1}M[r-1,r-1]
\end{array}\right]$$

For homomorphic multiplication, let $\boldsymbol{S} \in \mathbb{Z}_q^{r\times N}$ be the private key matrix, $\boldsymbol{R} \in \mathbb{Z}_q^{m\times N}$ and $\boldsymbol{E} \in \mathbb{Z}_q^{m\times N}$, we have:

$$\begin{aligned}
&\boldsymbol{SC_1C_2} \\
&= \boldsymbol{SC_1}\mathbf{G}^{-1}(\boldsymbol{C}_2) \\
&= \boldsymbol{E}_1\left(\boldsymbol{R}_1+\sum_{i=0}^{r-1}\sum_{j=0}^{r-1}M_1[i,j]\cdot\boldsymbol{R}_{\mathbf{1},(\boldsymbol{i},\boldsymbol{j})}\right)\mathbf{G}^{-1}(\boldsymbol{C}_2)+\boldsymbol{M_1SC}_2 \\
&= \boldsymbol{E}_1\left(\boldsymbol{R}_1+\sum_{i=0}^{r-1}\sum_{j=0}^{r-1}M_1[i,j]\cdot\boldsymbol{R}_{\mathbf{1},(\boldsymbol{i},\boldsymbol{j})}\right)\mathbf{G}^{-1}(\boldsymbol{C}_2) \\
&\quad+\boldsymbol{M_1E}_2\left(\boldsymbol{R}_2+\sum_{i=0}^{r-1}\sum_{j=0}^{r-1}M_2[i,j]\cdot\boldsymbol{R}_{\mathbf{1},(\boldsymbol{i},\boldsymbol{j})}\right)+\boldsymbol{M_1M_2SG}
\end{aligned} \tag{13}$$

let $\sum_{i=0}^{r-1}\sum_{j=0}^{r-1}|M_1[i,j]| = \rho_1$, $\sum_{i=0}^{r-1}\sum_{j=0}^{r-1}|M_2[i,j]| = \rho_2$. In *Lemma 3*, we know that if $N(1+\rho_1)\cdot\|\boldsymbol{E}_1\|_\infty + (1+\rho_2)\|\boldsymbol{M}_1\boldsymbol{E}_2\|_\infty < q/8$, the homomorphic multiplication can be decrypted correctly. From the above analysis, the size of noise is related not only to the size of matrices $\boldsymbol{E}_1$, and $\boldsymbol{M}_1$ $\boldsymbol{E}_2$ but also to ρ_1 and ρ_2.

In the case of the private-key encryption scheme, we have:

$$\begin{aligned}
\boldsymbol{SC}_1\boldsymbol{C}_2 &= \boldsymbol{SC}_1\mathbf{G}^{-1}(\boldsymbol{C}_2) \\
&= (\boldsymbol{E}_1+\boldsymbol{M}_1\boldsymbol{SG})\boldsymbol{G}^{-1}(\boldsymbol{C}_2) \\
&= \boldsymbol{E}_1\mathbf{G}^{-1}(\boldsymbol{C}_2)+\boldsymbol{M}_1\boldsymbol{SC}_2 \\
&= \boldsymbol{E}_1\mathbf{G}^{-1}(\boldsymbol{C}_2)+\boldsymbol{M}_1(\boldsymbol{E}_2+\boldsymbol{M}_2\boldsymbol{SG}) \\
&= \boldsymbol{E}_1\mathbf{G}^{-1}(\boldsymbol{C}_2)+\boldsymbol{M}_1\boldsymbol{E}_2+\boldsymbol{M}_1\boldsymbol{M}_2\boldsymbol{SG}
\end{aligned} \tag{14}$$

Similarly, it can decrypt the plaintext $\boldsymbol{M_1}$ $\boldsymbol{M_2}$ correctly when the absolute value of any element in $\boldsymbol{E_1}\mathbf{G^{-1}}(\boldsymbol{C_2})+\boldsymbol{M_1E_2}$ is less than $q/8$. Hence, the size of the noise is dependent on $\boldsymbol{E_1}$, $\boldsymbol{M_1E_2}$ and $\boldsymbol{C_2}$.

For the homomorphic addition of the private-key encryption scheme, if $\left\|\boldsymbol{E}^{(1)}+\boldsymbol{E}^{(2)}\right\|_\infty < q/8$, we can decrypt the plaintext $\boldsymbol{M}^{(1)}+\boldsymbol{M}^{(2)}$ correctly due to $\boldsymbol{S}(\boldsymbol{C_1}+\boldsymbol{C_2}) = E_1 + E_2 + (M_1+M_2)$ ***SG.*** Furthermore, the noise of ciphertext increases linearly with the number of ciphertexts. The same analysis also works on the public-key encryption scheme.

2) SECURITY ANALYSIS

We prove that the encryption scheme defined above is IND-CPA secure under the LWE hardness assumption.

Theorem 1: For any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that $Adv_{CPA}(\mathcal{A}) \le 2\cdot Adv_{LWE}(\mathcal{B})$.

Proof:

Game0(G_0): IND-CPA security experiment. According to the proposed scheme, challenger C first initializes the encryption scheme and then generates a public key $\boldsymbol{pk}:=\{(\boldsymbol{P}_{(i,j)}, \boldsymbol{B})\|1 \le i,j \le r\}$ and a private key $\boldsymbol{sk}:=\boldsymbol{S}$. The adversary $\mathcal{A}$obtains the public key of the scheme and selects two challenge plaintexts m_0 and m_1 from the plaintext space, then sends them to challenger C. Challenger C selects $b \in \{0,1\}$ at random and encrypts m_b using the public key. After that, the ciphertext is sent to the adversary $\mathcal{A}$. The adversary guesses the plaintext corresponding to the ciphertext and outputs b'. If $b'=b$, the adversary attacks successfully, and the advantage of adversary $\mathcal{A}$ is recorded as: $Adv_{CPA}[\mathcal{A}] = |\Pr[b=b' \text{ in game } G_0] - 1/2|$.

Game1(G1): In Game1, the public key $\boldsymbol{B} := \left[\frac{\bar{S}\boldsymbol{A}+\boldsymbol{E}}{\boldsymbol{A}}\right]$ which is used in Game0 is substituted by a uniform random value $\boldsymbol{B'} \xleftarrow{U} \mathbb{Z}_q^{(n+r)\times m}$. It is possible to verify that there exists an adversary $\mathcal{B}$ with the same running time as that of $\mathcal{A}$ such that $|\Pr[b=b' \text{ in game } G_1] - \Pr[b=b' \text{ in game } G_0]| \le Adv_{LWE}(\mathcal{B})$, because distinguishing $\boldsymbol{B}$ and $\boldsymbol{B'}$ for adversary $\mathcal{B}$ is as hard as solving **LWE** problem. Meanwhile, the other public key value $\boldsymbol{P}_{(i,j)}$ used in Game0 is also substituted by a uniform random value $\boldsymbol{P'}_{(i,j)} \xleftarrow{U} \mathbb{Z}_q^{(n+r)\times N}$. According to the game of *Definition 2* in Section II.C, it is possible to verify that there exists an adversary $\mathcal{B}$ with the same running time as that of $\mathcal{A}$ such that $|\Pr[b=b' \text{ in game } G_1] - \Pr[b=b' \text{ in game } G_0]| \le Adv_{LWE}(\mathcal{B})$, since $Adv_{Define2}[\mathcal{A}] = \left|\Pr[b=b'] - 1/2\right| = negl(\lambda)$.

Game2(G_2): In Game2, the value in the generation of the challenge ciphertext $\boldsymbol{C} := \boldsymbol{BR} + \sum_{i=1}^{r}\sum_{j=1}^{r}M[i,j]\cdot\boldsymbol{P}_{(\boldsymbol{i},\boldsymbol{j})}$ is substituted with uniform random elements in the matrix $\boldsymbol{C'} \xleftarrow{U} \mathbb{Z}_q^{(n+r)\times N}$ in Game1. The adversary distinguishes between $\boldsymbol{C'}$ and $\boldsymbol{C}$ is as difficult as the **LWE** problem, so there exists an adversary $\mathcal{B}$ with the same running time as that of $\mathcal{A}$ with $|\Pr[b=b' \text{ in game } G_2] - \Pr[b=b' \text{ in game } G_1]| \le Adv_{LWE}(\mathcal{B})$. Note that in Game2, the values in$\boldsymbol{C}$from the challenge ciphertext are independent of bit b; therefore, $\Pr[b=b' \text{ in game } G_2] = 1/2$.

In summary, $Adv_{CPA}(\mathcal{A}) \le 2\cdot Adv_{LWE}(\mathcal{B})$.

3) COMPLEXITY ANALYSIS

In this section, we analyze the computational complexity of the seIMC, including encryption, decryption and homomorphic computation.

For the public-key encryption algorithm (i.e., seIMC. **PubEnc()**), let $m = N = (n+r)\ell$; then, the computational complexity of the first term in (7) is $O((n+r)\times N\times m) = O((n+r)^3\ell^2)$, and the second term is $O(r\times r\times(n+r)\times N) = O(r^2(n+r)^2\ell)$. Hence, the total computation complexity of seIMC. **PubEnc()**is $O((n+r)^3\ell^2) + O(r^2(n+r)^2\ell)$.

For the private-key encryption algorithm (i.e., seIMC. **SecEnc()**), the computational complexity of (6) is $O(r\times n\times N) + O((n+r)^2\times N)$.

For the decryption algorithm (i.e., seIMC. **Dec()**), let $\boldsymbol{S} \in \mathbb{Z}_q^{r\times(n+r)}$and $\boldsymbol{C} \in \mathbb{Z}_q^{(n+r)\times N}$, the computational complexity of $\boldsymbol{H}=\boldsymbol{SC}$ is $O(r^2(n+r)\ell)$, since it only uses the first $r\ell$ columns of $\boldsymbol{C}$, instead of all the columns of $\boldsymbol{C}$. Then, we recover the plaintext $\boldsymbol{M}$ by calling the function **DecOneNum()** iteratively, and the computational complexity of this step is $O(r^2\ell)$. In summary, the total computational complexity of seIMC. **Dec()** is $O\left(r^2(n+r)\ell\right) + O\left(r^2\ell\right) \approx O\left(r^2(n+r)\ell\right)$.

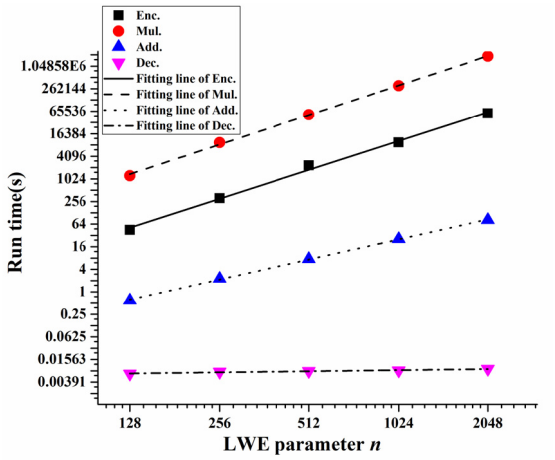


FIGURE 2. Time cost on the seIMC Private-key scheme with fixed $r = 8$ (Note that the y-axes is base 2.)

For the homomorphic operations of seIMC, we set the ciphertext $\boldsymbol{C} \in \mathbb{Z}_q^{(n+r)\times N}$ and $N = (n+r)\ell$, the computational complexity of seIMC. **Add()** is $O((n+r)^2\ell)$. The seIMC. **Mul()** has two operations: calculation of the matrix $\mathbf{G}^{-1}$ and matrix multiplication. The computational complexity of the former is $O((n+r)\ell^2)$, while that of the latter is $O((n+r)^3\ell^2)$. Therefore, the total computation complexity of seIMC. **Mul()** is $O\left((n+r)\,\ell^2\right) + O\left((n+r)^3\,\ell^2\right) \approx O\left((n+r)^3\,\ell^2\right)$.

IV. IMPLEMENTATION

A. EXPERIMENTAL SETTING

We conduct our implementations on the cloud server hosted at the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences. It is equipped with one Intel Xeon(R) E5-2680 2.4 GHz processor with 8 cores and 256 GB RAM. The operation system is Ubuntu 16.04. and the proposed seIMC scheme is written in Python 2.7. Furthermore, all the experiments are performed in sequence. To obtain a fair result, all the experiments are run 20 times independently, and the average value is taken as the final result.

For the LWE issue, we set a fixed parameter $q = 2^{30}$, the noise follows the sub-gaussian distribution with the variance $var = q/8m$, where $m = (n+r)\log q$, and $\ell = \log q = 30$.

B. VERIFICATION OF COMPLEXITY ANALYSIS

We first validate the complexity analysis of the proposed seIMC scheme with varying LEW parameter n. Fig. 2 shows the run time of each algorithm in the seIMC private-key scheme with varying LWE parameter n and fixed $r = 8$.

From Fig. 2, we can see that the time increase ratio of each algorithm in the seIMC private-key scheme is proportional to the dimension of LEW (i.e., n) with fixed r. In detail, the slope of the encryption algorithm is approximately $2.55 < 3$, which is consistent with the computational complexity of the encryption algorithm in the theoretical analysis (i.e., $O((n+r)^3\ell)$). Similarly, the growth ratio of the decryption algorithm conforms to the theoretical analysis results as well, since the slope of the decryption algorithm curve is 0.09 and the computational complexity of the decryption algorithm is $O(r^2(n+r)\ell)$. For the homomorphic addition and multiplication of seIMC, the growth ratios of each algorithm are 1.77 and 2.62, respectively, which are also satisfied by the computational result in the theoretical analysis (i.e., $O((n+r)^2\ell)$ and $O((n+r)^3\ell^2)$). Hence, the correctness of the computational complexity is verified, both from the perspective of the experiment and theory. Furthermore, Fig. 2 concludes that the homomorphic multiplication algorithm in seIMC is a time-consuming operation as it includes the computation of $\mathbf{G}^{-1}$ and matrix multiplication operations. Next, the encryption algorithm is nearly proportional to $O(n^3)$ with fixed r and ℓ. Then, the homomorphic addition algorithm is proportional to $O(n^{1.77})$ The run time of the decryption algorithm is the shortest in the seIMC scheme and is proportional to $O(n^{0.09})$.

C. IMPLEMENTATION EFFICIENCY COMPARISON

The implementation efficiency of seIMC is tested and compared with the state-of-art security matrix computation schemes (e.g., original GSW scheme [18], HAO scheme [27] and Jiang's scheme [25]). Due to the lack of a ready-made code base of GSW, we implement the GSW scheme proposed in [18] by using Python program. In detail, matrix encryption and decryption are performed by iteratively encrypting/decrypting the elements of the matrix, while the homomorphic operation is realized by transforming each element of the ciphertext matrix into a vector; then, it is calculated sequentially. The implementation efficiency of seIMC, HAO and GSW with different n and r values are shown in Table 1. Please note that the matrix operations are executed by using the Numpy library, which is a fundamental package for scientific computing in Python.

Table 1 shows that the proposed seIMC scheme outperforms the original GSW scheme and HAO scheme in both encryption and decryption. For example, when $n = 128$ and $r = 40$, the encryption time and decryption time of the original GSW scheme are 41382.4s and 369.7s, respectively, while those of the seIMC private-key scheme are only 0.96s and 0.0017s, respectively. Even for the seIMC public-key scheme, it takes only 7.62s and 0.16s on encryption and decryption, respectively. For HAO, it spends 6.897s to finish the encryption task and 0.588s to complete decryption. Hence, it demonstrates that the implementation efficiency of the seIMC scheme is significantly higher than that of the original GSW scheme and HAO scheme. Furthermore, the seIMC private-key scheme performs better than the public-key scheme in terms of run time.

To compare the efficiency of the homomorphic operations, we conduct a set of experiments with fixed $n = 32$ and varied r. The time taken for the homomorphic operations for seIMC, HAO, and GSW are shown in Table 2 and Table 3. Table 2 and Table 3 show that the time cost of the GSW scheme increases significantly as the data size increases.

TABLE 1. Time Cost Comparison of Encryption and Decryption with variant parameters.

Scheme	LWE parameter N	Size of matrix $r\times r$	Encryption/s	Decryption/s
seIMC private-key scheme	16	10×10	0.023	0.012
	32	20×20	0.090	0.049
	64	30×30	0.29	0.11
	128	**40×40**	**0.96**	**0.017**
	256	50×50	3.44	0.29
	512	60×60	11.04	0.48
seIMC public-key scheme	16	10×10	0.033	0.011
	32	20×20	0.17	0.046
	64	30×30	1.71	0.10
	128	**40×40**	**7.62**	**0.16**
	256	50×50	57.01	0.27
	512	60×60	3730.03	0.63
HAO scheme[27]	16	10×10	0.327	0.019
	32	20×20	1.245	0.089
	64	30×30	3.231	0.232
	128	**40×40**	**6.897**	**0.588**
	256	50×50	15.286	1.587
	512	60×60	34.260	3.275
GSW scheme [18]	16	10×10	41.62	0.78
	32	20×20	630.93	16.39
	64	30×30	5500.27	88.19
	128	**40×40**	**41382.43**	**369.74**

Taking homomorphic addition as an example, the encryption time of GSW is only 3.42s for the 2×2 matrix, whereas the encryption time of the 4×4 matrix is 13.48s. For the 8×8 matrix, the encryption time increases to 54.52s. When the size of the matrix increases to 128×128, it takes 13682.9s to perform the homomorphic addition operation. The same scenario also occurred for the homomorphic multiplication operation. For HAO, it takes 39.31s and 93.27s to execute homomorphic addition and multiplication operations, respectively, when the matrix size is 128×128, while it only costs approximately 0.8s and 0.82s for seIMC. Therefore, the proposed seIMC also outperforms the original GSW and HAO in terms of homomorphic operations. The reasons for this outperformance are as follows: 1) The original GSW suffers from the ciphertext space expansion issue, which encrypts each element (i.e., bit or integer) of the matrix into a ciphertext matrix. 2) The HAO scheme is only suitable for the bit matrix. It first needs to transform the integer matrix into a binary matrix, which leads to a low efficiency due to a large expansion rate of ciphertext and a sharp increase in the amount of homomorphic computation. Compared to the original GSW and HAO, the seIMC directly encrypts the whole plaintext matrix into a ciphertext matrix in the form of an integer. Therefore, the seIMC private-key scheme is more suitable for large-scale data processing under privacy protection in realistic scenarios.

Finally, we test the performance of seIMC compared with Jiang's scheme [25], it is a newly HE-based secure matrix computation scheme that includes a novel matrix encoding method and an efficient evaluation strategy for basic matrix

TABLE 2. Time taken comparison of homomorphic addition.

Matrix $(r\times r)$	seIMC/s	HAO/s	GSW/s
2×2	**0.04**	0.08	3.42
4×4	**0.04**	0.14	13.48
8×8	**0.05**	0.33	54.52
16×16	**0.07**	0.86	217.60
32×32	**0.13**	2.65	857.35
64×64	**0.20**	10.58	3450.85
128×128	**0.80**	39.31	13682.60

TABLE 3. Time taken comparison of homomorphic multiplication.

Matrix $r\times r$	seIMC/s	HAO/s	GSW/s
2×2	**0.04**	0.17	3.48
4×4	**0.04**	0.30	13.74
8×8	**0.05**	0.70	55.28
16×16	**0.07**	1.81	223.42
32×32	**0.13**	5.97	875.78
64×64	**0.30**	24.26	3520.63
128×128	**0.82**	93.27	14012.12

operations (e.g., matrix addition and multiplication). It has been demonstrated that the implementation of efficient homomorphic matrix multiplication in [25] outperforms other existing schemes, such as [22][23]. We set the security level of seIMC and Jiang to 80 in this experiment. The cyclotomic ring dimension of seIMC is chosen as $n = 450$ to achieve at least an 80-bit security level against the known attacks of the LWE problem, based on the estimator of Albrecht *et al.* [33]. The parameter settings of Jiang's scheme are the same as in [25]. The results are shown in Table 4. Clearly, the running times of Jiang's scheme are faster than those of the seIMC in terms of encryption and homomorphic addition operations. However, the performance of seIMC is better than Jiang's scheme in the following aspects: In the case of homomorphic multiplication, the execution efficiency of seIMC is higher than that of Jiang's scheme when dealing with matrices of size r greater than 32. Furthermore, Jiang's scheme fails to cope with the large-scale matrix (e.g., $r = 128$ or 256 in Table 4). 2). The same phenomena also occurred in the decryption algorithm. It means that the implementation efficiency of jiang's scheme is seriously declining with the matrix size increases, while seIMC still enjoys a high efficiency even if the matrix size expands. Therefore, it demonstrates that seIMC is more suitable for real applications with large-scale datasets.

V. APPLICATION TO SOCIAL NETWORK

In this section, we consider an untrusted cloud computing scenario to test seIMC on encrypted large-scale social network information. We propose a secure privacy data analysis solution in which data owners provide private social information to a public cloud and the cloud server offers a large-scale data analysis service to data owners who upload their encrypted data. In this instance, the cloud server should learn nothing

TABLE 4. Time cost of client and server with size.

Matrix r×r	seIMC				Jiang's method[25]			
	Enc/s	Homo.Add/s	Homo.Mult/s	Dec/s	Enc/s	Homo.Add/s	Homo.Mult/s	Dec/s
8×8	6.222	6.540	9.916	0.005	0.073	0.001	**2.98**	0.008
16×16	6.57	6.892	10.158	0.0186	0.0811	0.001	**6.558**	0.010
32×32	6.998	7.345	**10.639**	0.0768	0.09	0.001	15.592	0.0543
64×64	7.82	8.21	**12.287**	**0.312**	0.196	0.001	37.793	0.705
128×128	9.843	10.402	**15.824**	1.305	--	--	--	--
256×256	14.843	15.778	**24.055**	6.106	--	--	--	--

about the private information of the data owners. As the world's largest online social network, Facebook allows users to follow streams of posts generated by hundreds of their friends and acquaintances. Clearly, the user's friend circles belong in their privacy information.

In this experiment, the social network dataset is obtained from SNAP[1] (Stanford Network Analysis Project). It collects Facebook data and consists of 4,039 nodes and 88234 edges. Each node represents an individual, and the edge denotes the friend relation between two individuals. Note that the Facebook data have been anonymized by replacing the Facebook-internal ids for each user with a new value. Furthermore, we format the undirect graph of Facebook as the adjacency matrix $\boldsymbol{A}$. For instance, $A[i][j]$ represents whether user i and user j are friends. If $A[i][j]$=1, a friend relationship exists between user i and user j; otherwise, $A[i][j]$=0. The topology is described as follows:

Step 1: The client encrypts the adjacency matrix $\boldsymbol{A}$ to $E(\boldsymbol{A})$ by his private key using seIMC. **SecEnc**($\boldsymbol{A}$), and then the encrypted data $E(\boldsymbol{A})$ is uploaded to public clouds via the internet;

Step 2: The cloud server receives the encrypted data E($\boldsymbol{A}$) and calculates the left multiplication of $E(\boldsymbol{A})$ with K continuously by using seIMC. $\mathbf{Eval}_{\mathbf{mult}}$(E(**A**)). Then, it returns the encrypted result $E^k(\boldsymbol{A})$ to the client;

Step 3: The client receives the encrypted result $E^k(\boldsymbol{A})$ and uses the private key to decrypt it with seIMC. **Dec**($E^k(\boldsymbol{A})$). Finally, the client obtains $\boldsymbol{A}^k$, and each element of $\boldsymbol{A}^k$ represents the number of ways to recognize other users in the social network through k-times communication.

The implementation environment of seIMC is the same as section V. In detail, we set $q = 2^{30}$ and $n = 512$; the noise is a gaussian distribution with variance $q/8m$ (i.e., $var = q/8m$), $m = (n+r)\log q$, and $\ell = \log q = 30$. Besides, we also adopt the Numpy library to accelerate the computation speed of matrix operations. We take the first 1000 users in the dataset (i.e., $r = 1000$) and call the seIMC private-key encryption scheme. The estimated security parameter of the above setting is 90, based on the estimator of Albrecht *et al.* [33]. The time costs of encryption, decryption and homomorphic multiplication on client and cloud servers are shown in Table 5.

Instead of using the bootstrapping technology to flash the ciphertext and compress the noise expansion, we set a smaller noise, according to the number of multiplication layers, to ensure correct decryption. Table 5 shows that seIMC takes 78.25s and 34.95s to finish encryption and decryption, respectively. The left multiplication of encrypted social matrix E($\boldsymbol{A}$) (i.e., the total number of matrix $\boldsymbol{A}$ is 1,000,000) spends an average of 115.94 s on each homomorphic multiplication.

[1] http://snap.stanford.edu/data/

TABLE 5. Time cost of client and server with size.

Scheme	KeyGen/s	Enc/s	Eval/s			Dec/s
			k=2	k=3	k =4	
seIMC	97.96	78.25	115.76	116.33	115.73	34.95

The cloud server with the seIMC scheme can support not only the power operation, but also the homomorphic addition and subtraction operations for the statistical analysis. In addition, it is easy to modify the system to parallel execution to improve the operation efficiency. Therefore, it is feasible to apply the proposed integer matrix computation to protect and process sensitive matrix data in theory and practice.

VI. CONCLUSION

To address the secure and efficient data analysis under privacy protection, this paper proposes a GSW-based integer matrix computation scheme seIMC, which can guarantee the security of data processing and has a high computational efficiency. It includes the public-key and private-key encryption schemes. Furthermore, we also present the correctness and security proof of the proposed seIMC. Then, we evaluate the performance of the proposed seIMC in terms of efficiency, compared to the state-of-art schemes (i.e, GSW,HAO and Jiang's scheme). Finally, to solve the problem of the number of ways in which any two participants made friends throught k steps in an encrypted social network, we apply the seIMC private-key encryption scheme to the collected Facebook dataset with 1000 users. When the security level is 90, the serial encryption and decryption algorithm took approximately 1.3 minutes and 0.6 minutes, respectively; and homomorphic multiplication took approximately 1.9 minutes. Experiments showed that the proposed private-key encryption scheme can be effectively applied to privacy protection and secure data processing of integer matrix modulo q.

In future work, we plan to further expand the plaintext space and homomorphic operations for a wide range of applications in realistic scenarios.

REFERENCES

[1] A. Botta, "Integartion of cloud computing and Internet of Things: A survey," *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, Mar. 2016.

[2] E. Awad, S. Dsouza, R. Kim, J. Schulz, J. Henrich, A. Shariff, J.-F. Bonnefon, and I. Rahwan, "The moral machine experiment," *Nature*, vol. 563, no. 7729, pp. 59–64, Nov. 2018.

[3] Y. Wen, K. Zhang, and Z. Li, "A discriminative feature learning approach for deep face recognition," in *Proc. Comput. Vis. (ECCV)*, 2016, pp. 499–515.

[4] S. Singh, Y.-S. Jeong, and J. H. Park, "A survey on cloud computing security: Issues, threats, and solutions," *J. Netw. Comput. Appl.*, vol. 75, pp. 200–222, Nov. 2016.

[5] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Manual for using homomorphic encryption for bioinformatics," *Proc. IEEE*, vol. 105, no. 3, pp. 552–567, Mar. 2017.

[6] Y. Ma, L. Wu, X. Gu, J. He, and Z. Yang, "A secure face-verification scheme based on homomorphic encryption and deep neural networks," *IEEE Access*, vol. 5, pp. 16532–16538, 2017.

[7] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. 3rd ACM Workshop Cloud Comput. Secur. Workshop CCSW*, 2011, pp. 113–124.

[8] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–35, Sep. 2018.

[9] H. Takabi, J. B. D. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Secur. Privacy Mag.*, vol. 8, no. 6, pp. 24–31, Nov. 2010.

[10] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," *Found. Secure Comput.*, vol. 4, no. 11, pp. 169–180, 1978.

[11] D. Boneh, E. J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. 2th Theory Cryptogr. Conf.*, Cambridge, MA, USA, 2005, pp. 325–341.

[12] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. IT-31, no. 4, pp. 469–472, Jul. 1985.

[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 1999, pp. 223–238.

[14] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Symp. Theory Comput. STOC*, 2009, pp. 169–178.

[15] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM J. Comput.*, vol. 43, no. 2, pp. 831–871, 2014.

[16] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014.

[17] S. Halevi and V. Shoup, "Design and implementation of a homomorphic-encryption library," *IBM Res.*, vol. 6, pp. 12–15, Apr. 2013.

[18] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Annu. Cryptol. Conf.*, Santa Barbara, CA, USA, 2013, pp. 75–92.

[19] Y. Yin, L. Chen, Y. Xu, and J. Wan, "Location-aware service recommendation with enhanced probabilistic matrix factorization," *IEEE Access*, vol. 6, pp. 62815–62825, 2018.

[20] X. Chen, L. Wang, J. Qu, N.-N. Guan, and J.-Q. Li, "Predicting miRNA–disease association based on inductive matrix completion," *Bioinformatics*, vol. 34, pp. 4256–4265, Jun. 2018.

[21] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1273–1284, May 2014.

[22] D. Wu and J. Haven, "Using homomorphic encryption for large scale statistical analysis," FHE-SI-Report, Univ. Stanford, Stanford, CA, USA, Tech. Rep. TR-dwu4, 2012. [Online]. Available: http://cs.stanford.edu/~dwu4/FHE-SI_Report.pdf

[23] D. H. Duong, P. K. Mishra, and M. Yasuda, "Efficient secure matrix multiplication over LWE-based homomorphic encryption," *Tatra Mountains Math. Publications*, vol. 67, no. 1, pp. 69–83, Sep. 2016.

[24] P. K. Mishra, D. H. Duong, and M. Yasuda, "Enhancement for secure multiple matrix multiplications over ring-LWE homomorphic encryption," in *Proc. Inter'l Conf. Inf. Secur. Pract. Exper.*, Melbourne, VIC, Australia, 2017, pp. 320–330.

[25] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1209–1222.

[26] J. S. Alperin and C. Peikert, "Faster bootstrapping with polynomial error," in *Proc. Annu. Cryptol. Conf.*, Santa Barbara, CA, USA, 2014, pp. 297–314.

[27] R. Hiromasa, M. Abe, and T. Okamoto, "Packing messages and optimizing bootstrapping in GSW-FHE," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. E99. A, no. 1, pp. 73–82, 2016.

[28] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 34-1–34-40, 2009.

[29] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem," in *Proc. 41st Annu. ACM Symp. Symp. Theory Comput. STOC*, 2009, pp. 333–342.

[30] Z. Brakerski and V. Vaikuntanathan, "Lattice-based FHE as secure as PKE," in *Proc. 5th Conf. Innov. Theor. Comput. Sci. ITCS*, 2014, pp. 1–12.

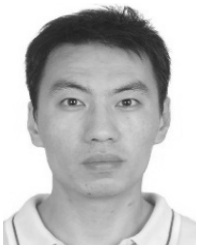
[31] D. Micciancio and C. Peikert, "Trapdoors for lattices: Simpler, tighter, faster, smaller," in *Proc. Annu. Inter'l Conf. Theory Appl. Cryptograph. Techn.*, Cambridge, U.K., 2012, pp. 700–718.

[32] Z. Li, S. D. Galbraith, and C. Ma, "Preventing adaptive key recovery attacks on the GSW levelled homomorphic encryption scheme," in *Proc. Inter'l Conf. Provable Secur.*, Nanjing, China, 2016, pp. 373–383.

[33] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, Jan. 2015.

[34] X. Shi, X. Luo, M. Shang, and L. Gu, "Long-term performance of collaborative filtering based recommenders in temporally evolving systems," *Neurocomputing*, vol. 267, pp. 635–643, Dec. 2017.

YANAN BAI was born in 1984. She is currently pursuing the Ph.D. degree with the University of Chinese Academy of Sciences, and studies in the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China. Her research interests include homomorphic encryption and applications, big data privacy protection, and cryptography theory.



XIAOYU SHI (Member, IEEE) received the B.S. degree in computer science from PLA Information Engineering University, Zhengzhou, China, in 2007, and the Ph.D. degree in computer science from the University of Electronic Science and Technology of China, Chengdu, China, in 2015. He joined the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China, in 2015, as an Associate Professor of computer science and engineering. His research interests include recommender systems, cloud computing, artificial intelligence, and big data applications.

WENYUAN WU was born in 1972. He is currently a Ph.D. Professor with the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China. His main research interests include cryptography theory, symbolic computation, zero error computation, and automated reasoning.

JINGWEI CHEN was born in 1984. He is currently a Ph.D. Associate Professor with the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China. His main research interests include symbol numerical mixing algorithm, lattice reduction algorithm, and lattice-based cryptography research.

YONG FENG was born in 1965. He is currently a Ph.D. Supervisor and a Professor with the Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China. He is also the Vice President of the Chongqing Society of electronics and the Chief Scientist of automatic reasoning and its application in high and new technology. His research interests include zero error computation in automatic reasoning, information security, and adaptive optical simulation.