IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received June 7, 2020, accepted July 20, 2020, date of publication July 27, 2020, date of current version August 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3011965

Securely and Efficiently Computing the Hermite Normal Form of Integer Matrices via Cloud Computing

WEI ZHAO[1], CHENGLIANG TIAN[1,2,3], WEIZHONG TIAN[4], AND YAN ZHANG[5], (Member, IEEE)

[1]College of Computer Science and Technology, Qingdao University, Qingdao 266071, China
[2]Business School, Qingdao University, Qingdao 266071, China
[3]State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[4]Department of Mathematical Sciences, Eastern New Mexico University, Portales, NM 88130, USA
[5]College of Electromechanical Engineering, Qingdao University of Science and Technology, Qingdao 266061, China

Corresponding author: Chengliang Tian (tcl0815@gmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61702294; in part by the National Development Foundation of Cryptography under Grant MMJJ20170126; and in part by the Open Research Project of the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, under Grant 2016-MS-23.

ABSTRACT The prevalence of cloud computing greatly promotes the traditional computing paradigm. With the assistance of a cloud, the light-weight device can achieve computation-intensive tasks which may not be done on its own. While, computing the Hermite Normal Form (HNF), which is a standard form of integer matrices, not only is inescapable when solving the linear system of equations over integers, but also has lots of applications in many other fields, such as integer programming and lattice-based cryptography. However, the fast blowup phenomenon of intermediate numbers makes the HNF computation time-consuming. In this paper, we initialize the study of the cloud-assisted HNF computation and design an efficient outsourcing algorithm that enables the resource-constrained client to securely delegate this heavy computation to a resource-abundant yet maybe untrusted cloud server. The main idea involved in our algorithm is a novel matrix encryption method based on random permutation, unimodular matrix transformation and triangular matrix transformation, which makes our algorithm protect the client's input/output information with the one-way notion and enable the client to detect the cloud's deception with the optimal probability 1. Besides, rigorous theoretical analysis and extensive experimental evaluation validate the efficiency and the practical performance of our design, and the substantial client-side savings are remarkable as the problem size increases.

INDEX TERMS Cloud computing, computation outsourcing, Hermite normal form, matrix transformation, privacy-preserving.

I. INTRODUCTION

The Hermite Normal Form (HNF), as a canonical form for integer matrices, has important applications in various computational problems in computer science. For example, in linear algebra just as reduced echelon form can be used to find solution vectors of a linear system over reals, the HNF is utilized to solve a linear system over integers [40]. In integer programming, the HNF has been employed to design an effective algorithm of finding the extreme points [19], and in lattice-based cryptography, the HNF has been used to improve the efficiency and security of lattice-based cryptosystems [32], [47]. Therefore, exploring the efficient algorithm of computing HNF is of great importance in the real world.

Given a full column rank matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$, the simplest algorithm of computing its HNF is the integer Gaussian elimination which triangularizes the matrix $\mathbf{A}$ with consecutive elementary matrices over integers. Intuitively, this algorithm only requires polynomial many arithmetic operations, and thus it could be expected to behave reasonably well concerning the size of the integers involved. Unfortunately, this is far from the practical case. Since the intermediate numbers explosion phenomenon occurs, the time overhead of this algorithm is exponential. For instance, Hafner and McCurley [17] give an example of a 20×20 integer matrix with entries

The associate editor coordinating the review of this manuscript and approving it for publication was Chin-Feng Lai.

between 0 and 10, and the numbers that are encountered during the execution of the algorithm can become as large as 10^{500} or even 10^{5011}. Subsequently, many researchers have tried to accelerate the HNF algorithm. The first polynomial-time algorithm for a square matrix $\mathbf{A}$ is given by Kannan and Bachem [21] with a time complexity of $\tilde{O}(n^{12}\log^2\|\mathbf{A}\|)$, where $\|\mathbf{A}\| = \max_{1\le i,j\le n}|a_{ij}|$ is the largest absolute value of entries of $\mathbf{A}$. Later, many improved algorithms with better complexity bounds are successively presented [8], [11], [20], [29], [33] and the currently best known algorithm is introduced by Storjohann and Labahn [44] with a provable asymptotic complexity of $\tilde{O}(mn^{\theta}\log\|\mathbf{A}\|)$, where $O(n^{\theta})$ denotes the upper bound for the number of arithmetic operations required to multiply two $n\times n$ integer matrices. It seems to be a feasible algorithm in practice. However, the integer matrices emerged in modern computer and engineering community, such as in machine learning, data mining, image processing, etc. usually involve hundreds of thousands of entries. Meanwhile, with the prevalence of 5G technology, more and more lightweight devices are connected to the internet. It is unrealistic for these resource-constrained entities to handle such large-scale matrices by themselves.

Cloud, as an unprecedented computing infrastructure, can provide ubiquitous network access to technology services, such as computing power, storage space, and databases, on a pay-as-you-go principle. As it brings the well-known benefits like low management cost, agile deployment, and elastic resource provision, outsourcing various computations to the cloud is intriguing for resource-constrained entities. However, every coin has two sides. The remote physical isolation between the resource-constrained client (outsourcer) and the cloud server makes this promising computing paradigm have to face some new security challenges [38], [49]. On the client side, the outsourcing computational task may contain sensitive information, such as personal secret keys, confidential image information, private property data, etc. Exposing this critical information could lead to the client's massive loss of life and asset. On the cloud side, due to internal hardware and software errors or external rantankerous economic incentives, the server may be lazy, curious, or even malicious, and thus the cloud could deviate the specified computation task, grab client's sensitive information or even deliberately forge malicious results returned to the client. Therefore, a sound and complete outsourcing computing algorithm should achieve the following security and performance guarantees: (1) Correctness. Any cloud server, as long as it executes the algorithm honestly, must be able to produce a result that can be successfully verified and decrypted by the client. (2) Input/output privacy. The cloud server cannot obtain sensitive information of the client's private data when executing the algorithm. (3) Verifiability. The client can verify whether the cloud server honestly executes the algorithm with a non-negligible probability. (4) Efficiency. The outsourcing algorithm must enable the client to achieve decent computational savings compared with that of performing the task without outsourcing.

A. RELATED WORK

Due to the above-mentioned tremendous benefits of cloud computing, designing secure and efficient outsourcing algorithms for various of computation-extensive tasks, such as large-scale linear algebraic operations [3], [14], modular exponentiations and modular inverse operations in cryptography [18], [27], [45], [56], large-scale graph operations [54], heavy computations in artificial intelligence (AI) and internet of things (IoT) [28], [51], [52], has become a popular topic. Out of which, the outsourcing of matrix-related operations is closely related with our work. For other computation outsourcing, one can refer to two comprehensive surveys presented by Shan *et al.* [42] and Yang *et al.* [50].

1) $\mathcal{MMC}$, $\mathcal{MIC}$, $\mathcal{MDC}$-OUTSOURCING

As three most fundamental matrix operations in linear algebra, matrix multiplication computation ($\mathcal{MMC}$), matrix inversion computation ($\mathcal{MIC}$) and matrix determinant computation ($\mathcal{MDC}$) are time-consuming when the size of the involved matrices is large, which is common in the era of big data. Atallah *et al.* [2] first investigated the $\mathcal{MMC}$ and $\mathcal{MIC}$ outsourcing and put forward privacy-preserving matrix disguise techniques based on random permutation. However, their algorithms are designed under the honest-but-curious single-server model and don't consider the verifiability. Benjamin and Atallah [3] initialized the study of verifiable $\mathcal{MMC}$ outsourcing strategy and, based on expensive homomorphic encryption schemes, they realized the secure outsourcing of MMC under the untrusted non-colluding two-server model. Later, Atallah and Frikken [1] improved the above-mentioned schemes and, by employing Shamir's secret sharing technique [41], designed a secure $\mathcal{MMC}$ outsourcing protocol under the malicious single-server model. Nevertheless, the redundancy check based on noise matrices makes their protocol suffer from poor efficiency. Subsequently, Mohassel [34] designed secure outsourcing algorithms for $\mathcal{MMC}$ and $\mathcal{MIC}$ by invoking existing block-box homomorphic encryption (HE) schemes [6], [16], [36]. Despite Mohassel's algorithm reduces client-side time cost to $O(n^2)$ in theory, it is also inefficient in practice due to the time-consuming HE operations. To overcome such expensive HE operations, Lei *et al.* [24]–[26] successively proposed three efficient outsourcing schemes for $\mathcal{MMC}$, $\mathcal{MIC}$ and $\mathcal{MDC}$. The key technique idea underlying their schemes is a concise matrix encryption/decryption method based on special sparse matrices. To amend the potential security issues incurred by simple sparse matrix transformations, recently, Zhang *et al.* [53] established a novel matrix encryption method based on consecutive sparse and unimodular matrix transformations, and applied this method to securely outsourcing $\mathcal{MMC}$, $\mathcal{MIC}$ and $\mathcal{MDC}$ over finite fields.

2) $\mathcal{LSLE}$-OUTSOURCING

Solving large-scale linear system of equations ($\mathcal{LSLE}$) is another ubiquitous linear algebraic operation in the computer science and engineering community. By utilizing a similar random permutation technique as employed in $\mathcal{MMC}$, $\mathcal{MDC}$-outsourcing, Atallah *et al.* [2] designed the first efficient $\mathcal{LSLE}$-outsourcing algorithm under the honest-but-curious single-server model, and thus the algorithm can not assure the result verification. Also, the simple permutation may leak the statistic information of entries in the coefficient matrix. Later, Wang *et al.* [48] revisited this problem and, based on the iterative method of solving $\mathcal{LSLE}$, proposed a privacy-preserving, cheating-immune outsourcing scheme. However, their scheme also suffers from low efficiency in practice due to the involved expensive HE scheme [36] and the multi-round interactions between the client and the server. Afterward, Chen *et al.* [7] successively adopted efficient random sparse-matrix transformations to outsourcing $\mathcal{LSLE}$ problem, which achieves high efficiency and optimal verifiability probability 1. However, just as mentioned in their paper, the simple sparse-matrix based blind technique can not provide strong enough security. Meanwhile, another efficient outsourcing algorithm given by Salinas *et al.* [39] was successfully attacked by Ding *et al.* [10]. Recently, to better balance the efficiency and security, Meng *et al.* [31], using the similar technique with reference [53], brought forward a publicly verifiable and efficiency/security-adjustable outsourcing algorithm for solving $\mathcal{LSLE}$ over some residue class ring $\mathbb{Z}_q$.

3) $\mathcal{MFC}$-OUTSOURCING

Various matrix factorization computations ($\mathcal{MFC}$) have extensive applications in data analysis, image processing, and AI, and the huge size of the matrices in the real world makes the $\mathcal{MFC}$ overloaded for recourse-constrained clients. Thus, outsourcing $\mathcal{MFC}$, such as non-negative matrix factorization (NMF), QR and LU factorizations, singular value decomposition (SVD), eigenvalue decomposition (EVD), etc. to a public cloud is also a hot topic. Based on the matrix permutation technique, Duan *et al.* [12] presented an efficient scheme to outsource the computation-intensive NMF in machine learning. However, some privacy breaches were revealed by Pan *et al.* [37]. Luo *et al.* [30] considered the outsourcing of the common large-scale QR and LU factorizations, and designed a privacy-preserving and verifiable outsourcing scheme by employing a low-complexity Householder transformation technique. Zhou and Li [55] realized the privacy-preserving and verifiable outsourcing of EVD and SVD of a matrix by utilizing random orthonormal transformation-based blind technique and Monte Carlo verification algorithm.

In all, most of the above matrix-related work is designed for matrices over reals. Whereas, because of the discrete structure of integers, the random matrix permutation techniques employed for real matrices are vulnerable to the greatest common divisor (GCD)-like algorithm's attack, and, meanwhile, operations on integer matrices are usually more expensive than that of matrices over reals. Therefore, designing secure and efficient outsourcing algorithms for widely applicable operations of integer matrices is of great importance.

B. OUR CONTRIBUTIONS

To help the resource-constrained client to fast fulfill the computation of HNF under the circumstance of cloud computing, this paper presents a cloud-assisted, privacy-preserving, and cheating-resistance outsourcing algorithm for this problem. That is, in our proposed algorithm, a local client can securely and efficiently obtain the HNF by delegating this overloaded computation task to a resource-abundant yet maybe malicious cloud server. As far as we know, this is the first outsourcing implementation of the HNF computation. More importantly, the main technique ingredient underlying our algorithm is a novel matrix encryption method based on permutation, unimodular transformation, and block upper triangular matrix, which makes the proposed algorithm has the following merits:

1) Our algorithm is designed under the fully malicious single-server model. Compared with the two (resp. multiple)-untrusted-server model which is common in the design of computation outsourcing schemes, this makes our algorithm more practical in the real world. Since the proposed algorithm only leverages one server and avoids the strong security assumption of two (resp. multiple) non-colluding servers.
2) The security of our algorithm has been shown to be robust. The encryption secret key in our algorithm is randomly chosen from a large enough key space and variable with the input matrix, which seems like one-time pad encryption, and we have presented a rigid argument on the one-way privacy of the client's input/output information. Moreover, the proposed algorithm can assure the client to detect the cloud server's fraud behavior with an optimal probability 1.
3) Our algorithm has been shown to be high-efficiency both in theory and practice. Our elaborate encryption technique skillfully transfers the complex HNF computation of a general matrix to the simple HNF computation of a block triangular matrix, which reduces the client's theoretical computation overhead from $\tilde{O}\left(mn^{\theta}\log\|\mathbf{A}\|\right)$ to $\tilde{O}\left((m^2n+m^{\theta})\log\|\mathbf{A}\|\right)$, where $2.3728639 \leq \theta \leq 3$, $\mathbf{A} \in \mathbb{Z}^{m\times n}$ is the input matrix and $\|\mathbf{A}\|$ denotes the largest absolute value of entries of $\mathbf{A}$. Extensive experimental analysis also indicates our algorithm's practical effectiveness and efficiency.

C. ROAD MAP

The road map of this paper is arranged as follows: we first show the system model and the associated security definitions in Section II, and introduce the preliminaries in Section III. Our main secure outsourcing algorithms are proposed in Section IV. Subsequently, Section V proves the correctness of our proposed algorithm, and analyzes its security and theoretical efficiency. The practical efficiency analysis and

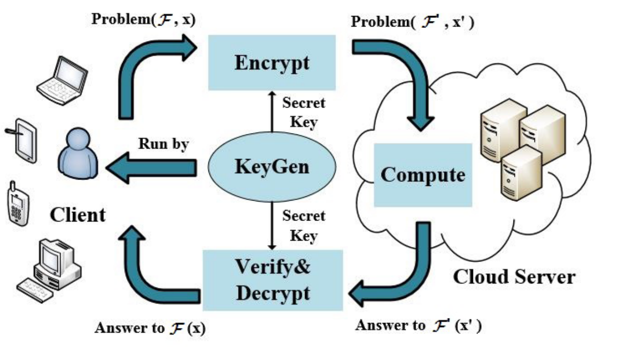

Problem($\mathcal{F}$, x)
Problem($\mathcal{F}'$, x')
Encrypt
Secret Key
Run by
KeyGen
Compute
Client
Secret Key
Verify& Decrypt
Cloud Server
Answer to $\mathcal{F}$ (x)
Answer to $\mathcal{F}'$ (x')

FIGURE 1. The system model.

performance evaluation are followed by Section VI. Finally, Section VII summarizes the paper.

II. SYSTEM MODEL AND SECURITY DEFINITIONS

A. SYSTEM MODEL

Generally, a typical secure computation outsourcing model can be seen as a special two-party computation system, which involves two entities with incoordinate computing capabilities: a resource-limited client C and a resource-abundant yet maybe untrusted cloud server S. As illustrated in FIGURE 1, the client C wants to leverage the computing power of S to achieve some overloaded computation problem $\mathcal{F}$ with an input x. To conceal the input information x, the client C firstly encrypts the problem $\mathcal{F}$ with the input x into another problem $\mathcal{F}'$ with corresponding encrypted input x' by utilizing a secret key SK. Then, C sends $(\mathcal{F}', x')$ to S. The cloud server S is specified to compute $y' = \mathcal{F}'(x')$ and send it back to C. After that, C checks the correctness of y'. If it is correct, by using the secret key SK, the client C decrypts y' to obtain the actual computational result $y = \mathcal{F}(x)$. Otherwise, the client C rejects the received result y'.

Formally, the general framework of a secure computation outsourcing algorithm is a four-tuple $\text{OAlg}_{\mathcal{F}} =$ (**KeyGen**, **Encrypt**, **ServerCom**, **Verify**&**Decrypt**) consisting of the following four probabilistic polynomial-time (PPT) sub-algorithms:

1) **KeyGen**$(\mathcal{F}, x, 1^{\kappa}) \rightarrow \{SK\}$: On Inputting a computation problem $\mathcal{F}$ with some input x, and a security parameter κ, the client C invokes the algorithm **KeyGen** to generate a secret key SK.
2) **Encrypt**$(\mathcal{F}, x, SK) \rightarrow \{\mathcal{F}', x'\}$: For the computation problem $\mathcal{F}$ with the input x, the client C performs the algorithm **Encrypt** to encrypt $\langle \mathcal{F}, x \rangle$ into another computation problem $\langle \mathcal{F}', x' \rangle$ by using his/her secret key SK, and then sends $\langle \mathcal{F}', x' \rangle$ to the cloud server S.
3) **ServerCom**$(\mathcal{F}', x') \rightarrow \{y'\}$: After receiving the computation problem $\mathcal{F}'$ with the blinded input x', the cloud server S performs the algorithm **ServerCom** to compute $y' = \mathcal{F}'(x')$ and sends it back to the client C.
4) **Verify**&**Decrypt**$(\mathcal{F}, y', SK) \rightarrow \{y, \perp\}$: After receiving the cloud server returned result y', the client C invokes the algorithm **Verify**&**Decrypt** to verify the correctness of y'. If y' is correct, the algorithm using SK decrypts y' into $y = \mathcal{F}(x)$ and outputs y. Otherwise, it outputs $\perp$.

B. THREAT MODELS

From the view of the users, the threats in a computation outsourcing system mainly come from the cloud server. According to different behaviors of the cloud server, there mainly exist three kinds of threat models in computing outsourcing system: the lazy single-server model, the semi-honest single-sever model and the fully malicious single-sever model.

Lazy Single-Server (LS) Model. In LS model, the cloud server is able to perform the protocol specification, but, for the sake of financial incentive, may return a random or an intermediate result.

Semi-honest Single-server (*SS*) Model. In *SS* model (also known as the "honest-but-curious" single-server model), the cloud server will fulfill the specified computation task honestly. However, it is curious about the client's private information, or even intentionally leaks the client's valuable information to outside attackers.

Fully Malicious Single-sever (*FMS*) Model. In *FMS* model, the cloud server can arbitrarily deviate from the specified computation task. It not only tries to pry into the client's private information, but also may return a forged result to fool the client.

Clearly, in the *LS* model, the client should be able to verify the correctness of the cloud server's returned results, and, in the *SS* model, the client has to protect the privacy of the input/output information. While, a secure outsourcing algorithm in the *FMS* model should enable the client to possess both of the two abilities. Hence, from the perspective of security, it is more meaningful to design a secure outsourcing algorithm in the *FMS* model than that of in the *LS* and *SS* models.

C. CORRECTNESS AND SECURITY DEFINITIONS

Based on our system and threat models, we give strict formalized definitions on the correctness and security requirements of a secure outsourcing algorithm.

1) CORRECTNESS

For some computation task $\mathcal{F}$, a secure computation outsourcing algorithm $\text{OAlg}_{\mathcal{F}}(\cdot)$ is correct if the client can achieve the desired value $y = \mathcal{F}(x)$ in case that the cloud server perform the delegated computation task honestly. Exactly

*Definition 1 (**Correctness**): A secure computation outsourcing algorithm* $\text{OAlg}_{\mathcal{F}}(\cdot)$ *is correct if, for any valid input* x*, the key generation algorithm produces* $\{SK\} \leftarrow$ **KeyGen**$(\mathcal{F}, x, 1^{\kappa})$ *such that, if* $\{\mathcal{F}', x'\} \leftarrow$ **Encrypt**$(\mathcal{F}, x, SK)$, $y' \leftarrow$ **ServerCom**$(\mathcal{F}', x')$ *and* $y' = \mathcal{F}'(x')$, *the algorithm* **Verify**&**Decrypt**$(\mathcal{F}, y', SK)$ *outputs* $y = \mathcal{F}(x)$.

2) INPUT/OUTPUT PRIVACY

Input/ouput privacy asks the secure computation outsourcing algorithm $\mathrm{OAlg}_{\mathcal{F}}(\cdot)$ to preserve the privacy of the client's input/output information as much as possible. That is, the less information an untrusted cloud server can infer about the input x (resp. the output $y = \mathcal{F}(x)$), the stronger input (resp. output) privacy the secure computation outsourcing algorithm achieves. Noteworthily, it employs secret-key encryption rather than public-key encryption in our system model, and the secret key is variable with the input infromation. Here, we argue the input (resp. output) privacy with one-way notion. Their definitions can be formalized with the following two experiments $Exp_{\mathcal{A}}^{Ipriv}[\mathcal{F}, 1^\kappa]$ and $Exp_{\mathcal{A}}^{Opriv}[\mathcal{F}, 1^\kappa]$.

```
Experiment $Exp_{\mathcal{A}}^{Ipriv}[\mathcal{F}, 1^\kappa]$
 *Query and response* :
 $x_0 = \sigma_{x_0} = \perp$.
 For $i = 1, \cdots, \ell = \mathrm{poly}(\kappa)$
   $x_i \leftarrow \mathcal{A}(\mathcal{F}, (x_j, \sigma_{x_j})_{0 \le j \le i-1})$.
   $SK_i \leftarrow \mathbf{KeyGen}(\mathcal{F}, x_i, 1^\kappa)$.
   $\sigma_{x_i} = (\mathcal{F}', x_i') \leftarrow \mathbf{Encrypt}(\mathcal{F}, SK_i, x_i)$.
 *Challenge* :
   $\hat{x} \leftarrow \mathrm{Domain}(\mathcal{F})$.
   $\hat{SK} \leftarrow \mathbf{KeyGen}(\mathcal{F}, \hat{x}, 1^\kappa)$.
   $\sigma_{\hat{x}} = (\mathcal{F}', \hat{x}') \leftarrow \mathbf{Encrypt}(\mathcal{F}, \hat{SK}, \hat{x})$.
   $\bar{x} \leftarrow \mathcal{A}(\mathcal{F}, (x_j, \sigma_{x_j})_{0 \le j \le \ell}, \sigma_{\hat{x}})$.
   if $\bar{x} = \hat{x}$ output '1';
   else output '0'.
```

In the above experiment $Exp_{\mathcal{A}}^{Ipriv}[\mathcal{F}, 1^\kappa]$, the adversary $\mathcal{A}$ adaptively chooses $\ell = \mathrm{poly}(\kappa)$ inputs $\{x_i\}_{1 \le i \le \ell}$ and queries their ciphertext $\{\sigma_{x_i}\}_{1 \le i \le \ell}$ by repeatedly invoking the algorithm **Encrypt**. In the *challenge* phase, the adversary $\mathcal{A}$ tries to calculate a value $\bar{x}$ according to the ciphertext $\sigma_{\hat{x}}$ of some challenge instance $\hat{x}$ and the information collected in the *query and response* phase. If $\bar{x} = \hat{x}$, the experiment outputs 1. Otherwise, the experiment outputs 0.

```
Experiment $Exp_{\mathcal{A}}^{Opriv}[\mathcal{F}, 1^\kappa]$
 *Query and response* :
 $x_0 = \sigma_{x_0} = \delta_0 = \perp$.
 For $i = 1, \cdots, \ell = \mathrm{poly}(\kappa)$
   $x_i \leftarrow \mathcal{A}(\mathcal{F}, (x_j, \sigma_{x_j}, \delta_j)_{0 \le j \le i-1})$.
   $SK_i \leftarrow \mathbf{KeyGen}(\mathcal{F}, x_i, 1^\kappa)$.
   $\sigma_{x_i} \leftarrow \mathbf{Encrypt}(\mathcal{F}, SK_i, x_i)$.
   $y_i' \leftarrow \mathcal{A}(\mathcal{F}, (x_j, \sigma_{x_j}, \delta_j)_{0 \le j \le i-1}, \sigma_{x_i})$.
   $\delta_i \leftarrow \mathbf{Verify\&Decrypt}(\mathcal{F}, SK_i, y_i')$.
 *Challenge* :
   $\hat{x} \leftarrow \mathrm{Domain}(\mathcal{F})$.
   $\hat{SK} \leftarrow \mathbf{KeyGen}(\mathcal{F}, \hat{x}, 1^\kappa)$.
   $\sigma_{\hat{x}} = (\mathcal{F}', \hat{x}') \leftarrow \mathbf{Encrypt}(\mathcal{F}, \hat{SK}, \hat{x})$.
   $\hat{y}' \leftarrow \mathbf{ServerCom}(\sigma_{\hat{x}})$.
   $\hat{y} \leftarrow \mathcal{A}(\mathcal{F}, (x_j, \sigma_{x_j}, \delta_j)_{0 \le j \le \ell}, \sigma_{\hat{x}}, \hat{y}')$.
   if $\hat{y} = \mathcal{F}(\hat{x})$, output '1';
   else output '0'.
```

In the *query and response* phase of the experiment $Exp_{\mathcal{A}}^{Opriv}[\mathcal{F}, 1^\kappa]$, given the oracle access to the algorithms **Encrypt** and **Verify&Decrypt**, the adversary $\mathcal{A}$ adaptively chooses $\ell = \mathrm{poly}(\kappa)$ three-tuples of $(x_i, \sigma_{x_i}, \delta_i)_{1 \le i \le \ell}$. In the *challenge* phase, given some challenge instance $\hat{x}$, the adversary $\mathcal{A}$ obtains $\sigma_{\hat{x}}$ and $\hat{y}'$ by invoking the algorithms **Encrypt** and **ServerCom**, as well as the collected information in the *query and response* phase. Then it tries to calculate a value $\hat{y}$. If $\hat{y} = \mathcal{F}(\hat{x})$, the experiment outputs 1. Otherwise, the experiment outputs 0.

*Definition 2 (**Input/output privacy**): A secure computation outsourcing algorithm* $\mathrm{OAlg}_{\mathcal{F}}(\cdot)$ *of some computation task* $\mathcal{F}$ *is input-private (resp. output-private) if, for any probabilistic polynomial-time adversary* $\mathcal{A}$*, the probability of the experiment* $Exp_{\mathcal{A}}^{Ipriv}[\mathcal{F}, 1^\kappa]$ *(resp.* $Exp_{\mathcal{A}}^{Opriv}[\mathcal{F}, 1^\kappa]$*) outputting* 1 *is negligible,* i.e.

$$\Pr[Exp_{\mathcal{A}}^{Ipriv}[\mathcal{F}, 1^\kappa] = 1] \le \mathtt{negli}(\kappa)$$
$$(resp. \Pr[Exp_{\mathcal{A}}^{Opriv}[\mathcal{F}, 1^\kappa] = 1] \le \mathtt{negli}(\kappa)),$$

where $\mathtt{negli}(\kappa)$ *is a negligible function of the security parameter.*

3) VERIFIABILITY

Informally, verifiability means that the outsourcing algorithm can enable the client to detect the an untrusted cloud server's fraudulent behavior with a non-negligible probability.

*Definition 3 ($(1-\beta)$-**Verifiable**): Given some computation task* $\mathcal{F}$*, a secure outsourcing algorithm* $\mathrm{OAlg}_{\mathcal{F}}(\cdot)$ *is* $(1-\beta)$*-verifiable if, for any valid input* x*, the secret key generating algorithm produces* $SK \leftarrow$ **KeyGen**$(\mathcal{F}, 1^\kappa)$ *such that, if* $(\mathcal{F}', x') \leftarrow$ **Encrypt**$(\mathcal{F}, x, SK)$*, and* $y' \leftarrow$ **ServerCom**$(\mathcal{F}', x')$*, the probability of* **Verify&Decrypt** $(\mathcal{F}, y', SK)$ *outputting* $y = \mathcal{F}(x)$ *satisfies*

$$\Pr[y \leftarrow \mathbf{Verify\&Decrypt}(\mathcal{F}, y', SK) \mid y' = \mathcal{F}'(x')] = 1,$$
$$\Pr[y \leftarrow \mathbf{Verify\&Decrypt}(\mathcal{F}, y', SK) \mid y' \ne \mathcal{F}'(x')] \le \beta.$$

4) EFFICIENCY

High-efficiency is the least requirement for a secure outsourcing computation algorithm, which can be defined as follows.

*Definition 4 (α-**Efficient**): Given some computation task* $\mathcal{F}$*, a secure outsourcing algorithm* $\mathrm{OAlg}_{\mathcal{F}}(\cdot)$ *is* α*-efficient if* $\frac{t_o}{t_c} \ge \alpha$*, where* t_o *denotes the client's time overhead of achieving the task on its own, and* t_c *is the local-client's time overhead of achieving the task by invoking the outsourcing algorithm* $\mathrm{OAlg}_{\mathcal{F}}(\cdot)$*.*

III. PRELIMINARIES

In this section, we introduce the notations and mathematical concepts used in this work.

A. NOTATIONS AND TERMINOLOGIES

We use bold upper (resp. lower) case letters to denote matrices (resp. vectors). For some $m \times n$ matrix $\mathbf{M}$, $\log \|\mathbf{M}\| = \log \max_{1 \le i \le m, 1 \le j \le n}\{|m_{ij}|\}$ denotes the bit length of the (absolute) maximum entry in $\mathbf{M}$. Throughout the paper, we use the notation log to denote logarithms to base 2. For any two real functions $f(x)$, $g(x)$, $f(x) = O(g(x))$ means $f(x) \le c_1 \cdot g(x)$

for some constant $c_1 > 0$ and any $x \geq x_0$, and $f(x) = \tilde{O}(g(x))$ if and only if $f(x) = O(g(x)\log^{c_2} g(x))$ for some constant $c_2 > 0$.

B. PERMUTATION MAPPING AND PERMUTATION MATRIX

Definition 5 (Permutation Mapping [5]): For any given finite set $S = \{1, 2, \cdots, n\}$, a permutation mapping defined on S is a bijection function $\pi : S \to S$, which is usually denoted as:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ \pi(1) & \pi(2) & \pi(3) & \cdots & \pi(n) \end{pmatrix},$$

where $\pi(1), \cdots, \pi(n)$ is some arrangement of $1, \cdots, n$.

Example 1: Let $n = 3$, then

$$\pi = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

is a permutation mapping defined on the set $\{1, 2, 3\}$.

Definition 6 (Permutation Matrix [4]): For any given finite set $S = \{1, 2, \cdots, n\}$ and a permutation mapping $\pi : S \to S$, the permutation matrix $\mathbf{P}_\pi$ induced by π is an $n \times n$ matrix with $\mathbf{P}_\pi(i, j) = \delta_{\pi(i),j}$ for any $1 \leq i, j \leq n$, where $\delta_{x,y}$ is the Kronecker delta function defined as

$$\delta_{x,y} = \begin{cases} 1, & x = y \\ 0, & x \neq y. \end{cases}$$

Example 2: Take the permutation mapping π as shown in example 1, the corresponding permutation matrix is

$$\mathbf{P}_\pi = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix},$$

since π is an even permutation, $\det(\mathbf{P}_\pi) = 1$.

C. UNIMODULAR MATRIX

Definition 7 (Unimodular Matrix [40]): A matrix $\mathbf{U} \in \mathbb{Z}^{n\times n}$ is unimodular if the absolute value of its determinant is 1, i.e. $|\det(\mathbf{U})| = 1$, where $\mathbb{Z}^{n\times n}$ denotes the set of $n \times n$ integer matrices.

Two simple properties of unimodular matrix are listed as the following two lemmas.

Lemma 1 ([35]): If $\mathbf{U} \in \mathbb{Z}^{n\times n}$ is unimodular, so is $\mathbf{U}^{-1}$. Particularly, if

$$\mathbf{U} = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \in \mathbb{Z}^{2\times 2},$$

then

$$\mathbf{U}^{-1} = \frac{1}{\det(\mathbf{U})} \times \begin{pmatrix} u_{22} & -u_{12} \\ -u_{21} & u_{11} \end{pmatrix} \in \mathbb{Z}^{2\times 2}.$$

Lemma 2: If $\mathbf{U} \in \mathbb{Z}^{n\times n}$ and $\mathbf{V} \in \mathbb{Z}^{n\times n}$ are unimodular, so is $\mathbf{U}' = \mathbf{U} \times \mathbf{V}$.

Example 3: Take $\mathbf{U} = \begin{pmatrix} 2 & -1 \\ 5 & -2 \end{pmatrix} \in \mathbb{Z}^{2\times 2}$ and $\mathbf{V} = \begin{pmatrix} 3 & -1 \\ 4 & -1 \end{pmatrix} \in \mathbb{Z}^{2\times 2}$. Clearly, $\det(\mathbf{U}) = 2\times(-2) - (-1)\times 5 = 1$ and $\det(\mathbf{U}) = 3 \times (-1) - (-1) \times 4 = 1$. They are both unimodular. It is easy to verify that $\mathbf{U}^{-1} = \begin{pmatrix} -2 & 1 \\ -5 & 2 \end{pmatrix} \in \mathbb{Z}^{2\times 2}$ and the product matrix $\mathbf{U}' = \mathbf{U} \times \mathbf{V} = \begin{pmatrix} 2 & -1 \\ 7 & -3 \end{pmatrix} \in \mathbb{Z}^{2\times 2}$ are also unimodular.

D. BLOCK MATRIX

A 2×2 block representation of an $(m + n) \times (m + n)$ matrix is

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}, \tag{1}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are $m \times m$, $m \times n$, $n \times m$ and $n \times n$ submatrices respectively.

Lemma 3: Assume $\mathbf{A}$ and $\mathbf{D}$ are invertible and $\mathbf{C} = \mathbf{0}$ in equation (1), then

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{D} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{D}^{-1} \\ \mathbf{0} & \mathbf{D}^{-1} \end{pmatrix}.$$

E. HERMITE NORMAL FORM

Definition 8 ([9]): An $m\times n$ matrix $\mathbf{H} = (h_{ij})$ with integer entries is in Hermite normal form (HNF) if there exists $r \leq m$ and a strictly monotonic increasing mapping f from $[r+1, n]$ to $[1, m]$ satisfying

- *For $r + 1 \leq j \leq m$, $h_{f(j),j} \geq 1$, $h_{i,j} = 0$ if $i > f(j)$ and $0 \leq h_{f(k),j} < h_{f(k),k}$ if $k < j$.*
- *The last $m - r$ rows of $\mathbf{H}$ are equal to zero.*

Specially, when $m = n$ and $f(k) = k$, $\mathbf{H}$ is in HNF if it satisfies the following conditions:

- $h_{i,i} > 0$ $(\forall 1 \leq i \leq n)$.
- $\mathbf{H}$ is an upper triangle matrix, *i.e.* $h_{i,j} = 0$ for $i \geq j$.
- $0 \leq h_{i,j} < h_{i,i}$ for $i > j$.

For example, the matrix

$$\mathbf{H} = \begin{pmatrix} 11 & 5 & 1 & 0 \\ 0 & 12 & 5 & 1 \\ 0 & 0 & 12 & 6 \\ 0 & 0 & 0 & 33 \end{pmatrix}$$

is in HNF. Notice that all the diagonal entries are greater than zero (condition 1); all off-diagonal entries in the lower half of the matrix are zeros (condition 2); and all off-diagonal entries in the upper half of the matrix are no less than zero and less than the corresponding value of the entry on the diagonal (condition 3).

An important property of HNF is provided as follows,

Lemma 4 ([9]): For any integer matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$, there exists an unique HNF matrix $\mathbf{H} \in \mathbb{Z}^{m\times n}$ and an unimodular matrix $\mathbf{U} \in \mathbb{Z}^{m\times m}$ such that $\mathbf{H} = \mathbf{U}\mathbf{A}$.

Example 4: Take

$$\mathbf{A} = \begin{pmatrix} 7 & -7 & 6 & -6 \\ 9 & -5 & -5 & -5 \\ 1 & 6 & 8 & 2 \\ -7 & -5 & -3 & -1 \end{pmatrix} \in \mathbb{Z}^{4\times 4}.$$

Its HNF matrix $\mathbf{H}$ and associated transformation matrix $\mathbf{U}$ are

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 338 \\ 0 & 1 & 0 & 24 \\ 0 & 0 & 1 & 401 \\ 0 & 0 & 0 & 922 \end{pmatrix}$$

and

$$\mathbf{U} = \begin{pmatrix} 102 & -274 & -361 & -302 \\ 7 & -19 & -25 & -21 \\ 121 & -325 & -428 & -358 \\ 278 & -747 & -984 & -823 \end{pmatrix}$$

satisfying $\mathbf{H} = \mathbf{U}\mathbf{A}$.

A simple corollary of Lemma 4 is

Lemma 5: Suppose $\mathbf{A}$ *and* $\mathbf{A}'$ *are two* $m \times n$ *integer matrices, and* $\mathbf{A}' = \mathbf{U}'\mathbf{A}$ *for some unimodular matrix* $\mathbf{U}' \in \mathbb{Z}^{m\times m}$*, then* $\mathbf{A}'$ *and* $\mathbf{A}$ *share the same HNF.*

Proof: We assume that $\mathbf{H}$ is the HNF matrix of $\mathbf{A}$. By Lemma 4, there exists an unimodular matrix $\mathbf{U}$ such that $\mathbf{H} = \mathbf{U}\mathbf{A}$. Since $\mathbf{A} = \mathbf{U}'^{-1}\mathbf{A}'$, we have

$$\mathbf{H} = \mathbf{U}\mathbf{U}'^{-1}\mathbf{A}'.$$

According to Lemma 2 and Lemma 4, $\mathbf{U}\mathbf{U}'^{-1}$ is unimodular and the HNF matrix of $\mathbf{A}'$ is unique. Therefore, the HNF matrix of $\mathbf{A}'$ is $\mathbf{H}$. *I.e.* $\mathbf{A}'$ and $\mathbf{A}$ share the same HNF matrix $\mathbf{H}$. □

On the algorithmic side, Storjohann and Labahn [44] developed the currently provable and fastest algorithm of computing the HNF.

Lemma 6 ([44]): There exists a deterministic algorithm that, on inputting an $m \times n$ *integer matrix* $\mathbf{A}$ *of rank* n*, outputs the HNF matrix* $\mathbf{H} \in \mathbb{Z}^{m\times n}$ *and an unimodular matrix* $\mathbf{U} \in \mathbb{Z}^{m\times m}$ *such that* $\mathbf{H} = \mathbf{U}\mathbf{A}$ *in time* $\tilde{O}(mn^{\theta-1}M(n\log\|\mathbf{A}\|, n\log\|\mathbf{A}\|))$*. Moreover, the total size of* $\mathbf{H}$ *(the summation of the bit lengths of the individual entries of* $\mathbf{H}$*) is on the order of* $\tilde{O}(mn\log\|\mathbf{A}\|)$ *bits. Entries in* $\mathbf{U}$ *and* $\mathbf{U}^{-1}$ *will be bounded in length by* $\tilde{O}(n\log\|\mathbf{A}\|)$ *bits and by* $\tilde{O}(\log\|\mathbf{A}\| + \log n)$ *bits respectively.*

Here, in Lemma 6, $M(x, y)$ denotes the asymptotic upper bound for the number of bit operations required to multiply two number with x, y bits respectively. Clearly, $M(x, y) = xy$ for standard operations and $M(x, y) = \max\{x, y\}\log\max\{x, y\} = \tilde{O}(\max\{x, y\})$ for asymptotically fastest algorithm [22]. $O(n^\theta)$ stands in for asymptotically upper bound for the number of arithmetic operations required to multiply two $n \times n$ integer matrices. $\theta = 3$ for practical standard operations and $\theta = 2.3728639$ for the asymptotically fastest algorithm [15].

Specially, for a triangular integer matrix, there exists a more efficient algorithm to compute the HNF.

Lemma 7 ([43]): There exists a deterministic algorithm that, on inputting an $n \times n$ *upper triangular integer matrix* $\mathbf{B}$*, outputs the HNF of* $\mathbf{B}$ *in time* $O(n^2M(\log D, \log D))$*, where* $D = \det(\mathbf{B})$.

IV. OUTSOURCING ALGORITHM OF HNF

A. A HIGH LEVEL DESCRIPTION

Give an $m\times n$ $(m \geq n)$ integer matrix $\mathbf{A}$ of rank n, we intend to compute its HNF with the assistance of a resource-abundant yet maybe untrusted cloud server.

To protect the privacy of $\mathbf{A}$, a common method is to take advantage of the classic permutation-substitution encryption technique. That is. we can multiply the original matrix $\mathbf{A}$ by a permutation matrix $\mathbf{P}$ and a unimodular matrix $\mathbf{U}$ on the left side. Although this simple transformation can blind the position and value information of the entries in $\mathbf{A}$, by Lemma 5, it can not ensure the output privacy. *I.e.*, the cloud server can obtain the HNF of $\mathbf{A}$ by computing the HNF of $\mathbf{A}' = \mathbf{UPA}$. To amend this flaw, we must figure out a new encryption method with the following two properties: (1) It not only can blind the input matrix $\mathbf{A}$, but also is able to conceal its HNF. (2) The method can ensure the local client to recover the HNF of $\mathbf{A}$ from the cloud server returned results correctly and efficiently. Based on HNF's definition (Definition 8) and its property (Lemma 5), we adapt the original idea with multiplying the matrix $\mathbf{A}$ by a permutation matrix $\mathbf{P}$, a unimodular matrix $\mathbf{U}$ on the left side and a block upper triangular matrix $\mathbf{R}$ on the right side simultaneously. Namely, the client delegates the computation task of computing the HNF of

$$\mathbf{A}' = \mathbf{UPAR} \tag{2}$$

to the cloud server. Let $\mathbf{H}'$ denote the HNF of $\mathbf{A}'$. Then, by Lemma 5, there exists a unimodular matrix $\mathbf{U}'$ such that

$$\mathbf{H}' = \mathbf{U}'\mathbf{A}'. \tag{3}$$

Combing equation (2) with equation (3), we have $\mathbf{A} = \mathbf{P}^{-1}\mathbf{U}^{-1}(\mathbf{U}')^{-1}\mathbf{H}'\mathbf{R}^{-1}$. Therefore, once receiving the cloud server returned result $\mathbf{H}'$, the client can recover the HNF of $\mathbf{A}$ by computing the HNF of a block upper triangular matrix $\mathbf{H}'\mathbf{R}^{-1}$. Since computing the HNF of a block upper triangular matrix is easy, this makes our algorithm efficient. Meanwhile, the randomness of the matrices $\mathbf{P}$, $\mathbf{U}$ and $\mathbf{R}$ guarantees the algorithm's input/output privacy.

B. THE DETAILS OF OUTSOURCING ALGORITHM

Concretely, our HNF-outsourcing algorithm $\text{OAlg}_{HNF}(\cdot) =$ (**KeyGen**, **Encrypt**, **ServerCom**, **Verify**&**Decrypt**) consists of four subalgorithms.

1) KEY GENERATION ALGORITHM

On input an $m \times n$ integer matrix $\mathbf{A}$ and a security parameter λ, the key generation algorithm **KeyGen** outputs a random secret key $sk = (\mathbf{R}, \mathbf{P}, \mathbf{U})$, where $\mathbf{R} \in \mathbb{Z}^{n\times n}$ is a random upper triangular and nonsingular matrix generated by Algorithm 1, $\mathbf{P} \in \{0, 1\}^{m\times m}$ is a random permutation matrix generated by Algorithm 2, and $\mathbf{U} \in \mathbf{Z}^{m\times m}$ is a block-diagonal unimodular matrix generated by Algorithm 3.

Algorithm 1 R(A, 1^λ)

```
Input: A matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$ of rank $n$ and a security parameter $\lambda$.
Output: A random block triangular matrix $\mathbf{R} \in \mathbb{Z}^{n\times n}$
1: For $i = 1$ to $n$
2:     Choose $r_{i,i}$, $r_{i,i+1}$ from $\mathbb{Z} \cap (-2^\lambda, 2^\lambda)$ uniformly at random such that $r_{i,i} \neq 0$
3:     Choose $r_{21}$ from $\mathbb{Z} \cap (-2^\lambda, 2^\lambda)$ uniformly at random such that $|r_{11}r_{22} - r_{21}r_{12}| \neq 0$
4:     For $j = 1$ to $n$ and $(i, j) \neq (2, 1), (i, i), (i, i + 1)$
5:         $r_{ij} = 0$
6: Construct
       $$\mathbf{R}_1 = \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix}, \mathbf{R}_2 = \begin{pmatrix} 0 & 0 \cdots 0 \\ r_{23} & 0 \cdots 0 \end{pmatrix},$$
       $$\mathbf{O} = (r_{ij})_{3\le i\le n, 1\le j\le 2} = \begin{pmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix},$$
   and
       $$\mathbf{R}_3 = (r_{ij})_{3\le i,j\le n} = \begin{pmatrix} r_{33} & r_{34} & & & \\ & r_{44} & r_{45} & & \\ & & \ddots & & \\ & & & r_{n-1\,n-1} & r_{n-1\,n} \\ & & & & r_{nn} \end{pmatrix}$$
   is a sparse upper-triangular matrix
7: Return $\mathbf{R} = \begin{pmatrix} \mathbf{R}_1 & \mathbf{R}_2 \\ \mathbf{O} & \mathbf{R}_3 \end{pmatrix}$
```

Algorithm 2 P(A, 1^λ)

```
Input: A matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$ of rank $n$ and a security parameter $\lambda$.
Output: A random permutation matrix $\mathbf{P} \in \mathbb{Z}^{m\times m}$
1: Set $\pi = \mathbf{I}_m$(identical permutation)
2: for $i = m$ to 2
3:     Set $j$ to be a random integer with $1 \le j \le i$
4:     Swap $\pi[j]$ and $\pi[i]$
5: for $i = 1$ to $m$
6:     for $j = 1$ to $m$
7:         $p_{ij} = \delta_{\pi(i),j}$
8: Return $\mathbf{P} = (p_{ij})_{1\le i,j\le m}$
```

2) CLIENT ENCRYPTION ALGORITHM

In this stage, the client utilizes the secret key $sk = (\mathbf{R}, \mathbf{P}, \mathbf{U})$ to encrypt the original matrix $\mathbf{A}$ by computing $\mathbf{A}' = \mathbf{UPAR}$ and sends $\mathbf{A}'$ to the cloud server.

3) SERVER COMPUTING ALGORITHM

After receiving the encrypted matrix $\mathbf{A}'$ from the client, the cloud server are inquired to compute $(\mathbf{H}', \mathbf{V}')$ such that $\mathbf{H}' \in \mathbb{Z}^{m\times n}$ is the HNF of $\mathbf{A}'$ and $\mathbf{V}' \in \mathbb{Z}^{m\times m}$ is an unimodular

Algorithm 3 U(A, 1^λ)

```
Input: A matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$ of rank $n$ and a security parameter $\lambda$.
Output: A block-diagonal unimodular matrix $\mathbf{U} \in \mathbb{Z}^{m\times m}$
1: For $i = 1$ to $\lceil m/2 \rceil$
2:     Choose two coprime random integers $u_{11}^{(i)}, u_{12}^{(i)} \in \mathbb{Z} \cap (-2^\lambda, 2^\lambda)$
3:     Perform the extended Euclidean algorithm to $u_{11}^{(i)}, u_{12}^{(i)}$ and return two integers $u_{21}^{(i)}, u_{22}^{(i)} \in \mathbb{Z} \cap (-2^\lambda, 2^\lambda)$ such that $|u_{11}^{(i)}u_{22}^{(i)} - u_{21}^{(i)}u_{12}^{(i)}| = 1$.
4:     Construct $\mathbf{U}_i = \begin{pmatrix} u_{11}^{(i)} & u_{12}^{(i)} \\ u_{21}^{(i)} & u_{22}^{(i)} \end{pmatrix}$
5: if $m\%2 == 0$
6:     Return $\mathbf{U} = \mathrm{diag}(\mathbf{U}_1, \cdots, \mathbf{U}_{\lceil \frac{m}{2} \rceil})$
7: else
8:     Return
       $$\mathbf{U} = \mathrm{diag}(\mathbf{U}_1, \cdots, \mathbf{U}_{\frac{m-1}{2}}, 1) \cdot \mathrm{diag}(1, \cdots, 1, \mathbf{U}_{\lceil m/2 \rceil})$$
```

matrix satisfying $\mathbf{V}'\mathbf{H}' = \mathbf{A}'$. Then it returns $(\mathbf{H}', \mathbf{V}')$ to the client.

4) CLIENT VERIFICATION AND DECRYPTION ALGORITHM

Once receiving the cloud server returned result $(\mathbf{H}', \mathbf{V}')$, the client first checks whether $\mathbf{H}'$ is in HNF and $\mathbf{V}' \in \mathbb{Z}^{m\times m}$. If not, the client rejects the result. Else, the client further verifies whether $\mathbf{V}'\mathbf{H}' = \mathbf{A}'$ and $|\det(\mathbf{V}')| = 1$. If it holds, the client computes $\mathbf{B} = \mathbf{H}'\mathbf{R}^{-1}$ and outputs its HNF matrix $\mathbf{H}$. Else, it rejects the result. The detail description is shown in Algorithm 4.

Algorithm 4 Verify&Decrypt

```
Input: Matrices $\mathbf{H}'$, $\mathbf{V}'$, $\mathbf{A}'$ and $\mathbf{R}$.
Output: An HNF matrix $\mathbf{H}$ or $\perp$.
1: If $\mathbf{H}'$ is not in HNF or $\mathbf{V}' \notin \mathbb{Z}^{m\times m}$
2:     Return $\perp$
3: Else
4:     If $\mathbf{V}'\mathbf{H}' = \mathbf{A}'$ and $|\det(\mathbf{V}')| = 1$
5:         The client computes $\mathbf{R}^{-1}$
6:         The client computes $\mathbf{B} = \mathbf{H}'\mathbf{R}^{-1}$
7:         If $\mathbf{B} \in \mathbb{Z}^{m\times n}$
8:             The client computes the HNF matrix $\mathbf{H}$ of $\mathbf{B}$
9:             Return $\mathbf{H}$
10:        Else
11:            Return $\perp$
12:    Else
13:        Return $\perp$
```

Now, we further illustrate our algorithm with a toy example.

Example 5: Take the matrix

$$\mathbf{A} = \begin{pmatrix} -1 & 0 & -14 & 9 & 13 \\ -5 & 6 & 12 & 15 & 6 \\ 8 & -10 & -8 & -3 & 15 \\ 4 & 14 & 3 & -9 & 4 \\ 13 & 16 & -12 & -4 & 11 \\ -4 & -12 & 14 & 13 & -11 \end{pmatrix} \in \mathbb{Z}^{6\times 5}.$$

The proposed algorithm $\mathrm{OAlg}_{HNF}(\mathbf{A})$ goes as follows: (1) The client randomly generates a block triangular matrix

$$\mathbf{R} = \begin{pmatrix} 4 & -2 & 0 & 0 & 0 \\ 2 & 4 & -3 & 0 & 0 \\ 0 & 0 & 2 & -3 & 0 \\ 0 & 0 & 0 & 3 & -2 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix},$$

and a permutation matrix

$$\mathbf{P} = \begin{pmatrix} 0&0&0&0&0&1 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&0&1&0 \\ 1&0&0&0&0&0 \\ 0&0&0&1&0&0 \end{pmatrix}.$$

Then, the client randomly chooses three 2×2 unimodular matrices

$$\mathbf{U_1} = \begin{pmatrix} 2 & -3 \\ -1 & 2 \end{pmatrix}, \mathbf{U_2} = \begin{pmatrix} 1 & -2 \\ -1 & 1 \end{pmatrix}, \mathbf{U_3} = \begin{pmatrix} -1 & -1 \\ 2 & 1 \end{pmatrix},$$

and constructs the secret unimodular matrix

$$\mathbf{U} = \begin{pmatrix} 2 & -3 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 2 & 1 \end{pmatrix}.$$

In all, the client's secret key $sk = (\mathbf{R}, \mathbf{P}, \mathbf{U})$.

(2) Utilizing the secret key sk, the client encrypts the input matrix $\mathbf{A}$ into

$$\mathbf{A}' = \mathbf{UPAR} = \begin{pmatrix} -56 & -182 & 110 & -33 & -82 \\ 24 & 108 & -52 & 21 & 35 \\ -156 & -132 & 158 & -33 & -31 \\ 72 & 94 & -86 & 9 & -10 \\ -40 & -50 & 64 & -33 & -51 \\ 36 & 52 & -92 & 102 & 72 \end{pmatrix},$$

and sends the ciphertext matrix $\mathbf{A}'$ to the cloud server.

(3) After receiving the encrypted matrix $\mathbf{A}'$, the cloud server computes its HNF matrix $\mathbf{H}'$ and associated unimodular transformation matrix $\mathbf{V}'$:

$$\mathbf{H}' = \begin{pmatrix} 4 & 8 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\mathbf{V}' = \begin{pmatrix} -14 & -7 & 55 & -11 & -42 & 10 \\ 6 & 6 & -26 & 7 & 18 & -6 \\ -39 & 18 & 79 & -11 & -33 & -11 \\ 18 & -5 & -43 & 3 & 10 & 9 \\ -10 & 3 & 32 & -11 & -24 & 3 \\ 9 & -2 & -46 & 34 & 28 & -6 \end{pmatrix}.$$

and returns them to the client.

(4) Once receiving the returned results from cloud server, the client first verifies the correctness of the returned results. *I.e.* The client checks whether $\mathbf{H}'$ is in HNF and $\mathbf{A}' = \mathbf{V}'\mathbf{H}'$. If they pass the verification, the client confirms the determinant of matrix $\mathbf{V}'$. Since $\det(\mathbf{V}') = -1$, the client computes

$$\mathbf{B} = \mathbf{H}' \times \mathbf{R}^{-1} = \begin{pmatrix} 0 & 2 & 3 & 3 & 2 \\ -1 & 2 & 3 & 3 & 2 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 21 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

and recovers the actual result by computing the HNF of the matrix $\mathbf{B}$:

$$\mathbf{H} = \begin{pmatrix} 1&0&0&0&0 \\ 0&2&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \\ 0&0&0&0&0 \end{pmatrix}.$$

V. CORRECTNESS AND SECURITY ANALYSIS

In this section, we will give a rigorous analysis on the correctness and the security of our proposed algorithm according to the definitions modeled in section II-C.

A. CORRECTNESS

Here, correctness means that the client can obtain the HNF of any valid input matrix correctly in case that the cloud server performs the specified computation task honestly.

Theorem 1: For any large-scale input matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$ *of rank n, the proposed algorithm* $\mathrm{OAlg}_{HNF}(\mathbf{A})$ *is correct according to Definition 1.*

Proof: Based on the encryption algorithm, the ciphertext matrix $\mathbf{A}' = \mathbf{UPAR}$. If the cloud server is honest, $\mathbf{V}'\mathbf{H}' = \mathbf{A}'$ and $\mathbf{V}'$ is unimodular. That is, $\mathbf{V}'\mathbf{H}' = \mathbf{UPAR}$. Therefore,

$$\mathbf{P}^{-1}\mathbf{U}^{-1}\mathbf{V}'\mathbf{H}'\mathbf{R}^{-1} = \mathbf{A}.$$

Since $\mathbf{P}$ and $\mathbf{U}$ are also unimodular, $\mathbf{B} = \mathbf{H}'\mathbf{R}^{-1} = (\mathbf{V}')^{-1}\mathbf{UPA}$ is an integer matrix. By Lemma 5, $\mathbf{A}$ shares the same HNF matrix with $\mathbf{B}$. Consequently, the output matrix $\mathbf{H}$

is the HNF of $\mathbf{A}$, *i.e.* the proposed algorithm $\mathrm{OAlg}_{HNF}(\cdot)$ is correct. $\square$

B. INPUT/OUTPUT PRIVACY

Now, we argue the one-way input/output privacy of the proposed algorithm.

Theorem 2: For any large-scale input matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$ *of rank n, the proposed algorithm* $\mathrm{OAlg}_{HNF}(\mathbf{A})$ *fulfills the input/output privacy according to Definition 2.*

Proof: (1) Input privacy. In the experiment $Exp_{\mathcal{A}}^{Ipriv}[\mathcal{F}, 1^{\kappa}]$, the computation task $\mathcal{F}$ represents the HNF computation of $\mathbf{A}$, and $\kappa = mn \log \|\mathbf{A}\|$ is the bit-length of the input information. In the *Query and response* phase, the adversary $\mathcal{A}$ can adaptively choose $(x_i, \sigma_{x_i}) = (\mathbf{A}_i, \mathbf{A}_i')$ for $1 \le i \le \ell$. In the *Challenge* phase, given the ciphertext matrix $\mathbf{A}'$ of some challenge instance $\mathbf{A}$, the adversary tries to recover $\mathbf{A}$.

Now we analyze the probability that the adversary can successfully obtain $\mathbf{A}$. According to the client encryption algorithm, $\mathbf{A}' = \mathbf{UPAR}$, *i.e.* $\mathbf{A} = \mathbf{P}^{-1}\mathbf{U}^{-1}\mathbf{A}'\mathbf{R}^{-1}$. Let $\mathbf{X} = \mathbf{P}^{-1}\mathbf{U}^{-1}\mathbf{A}'$. Then $Rank(\mathbf{X}) = Rank(\mathbf{A}') = Rank(\mathbf{A}) = n$. Hence the mapping $f(\mathbf{r}) = \mathbf{Xr}$ is an injective function from $\mathbb{R}^n$ to $\mathbb{R}^m$. That is, the input matrix $\mathbf{A}$ varies as the variation of the secret matrix $\mathbf{R}$. Since the entries in $\mathbf{R}$ are chosen uniformly at random from $\mathbb{Z} \cap (-2^{\lambda}, 2^{\lambda})$, the probability that the adversary can obtain the correct $\mathbf{A}$ is

$$\begin{aligned}&\frac{1}{|\{\mathbf{A} \mid \mathbf{A} = \mathbf{P}^{-1}\mathbf{U}^{-1}\mathbf{A}'\mathbf{R}^{-1}\}|}\\ &\quad\le \frac{1}{|\{\mathbf{R} \mid \mathbf{R} \text{ is constructed as in Alg.1}\}|}\\ &\quad\le \frac{1}{(2(2^{\lambda}-1))^{n+1}(2\cdot 2^{\lambda}-1)^{n-1}},\end{aligned}$$

which obviously is negligible.

(2) Output privacy. Similarly, In the *Query and response* phase of experiment $Exp_{\mathcal{A}}^{Opriv}[\mathcal{F}, 1^{\kappa}]$, the adversary $\mathcal{A}$ can adaptively choose $(x_i, \sigma_{x_i}, \delta_i) = (\mathbf{A}_i, \mathbf{A}_i', \mathbf{H}_i)$ (or $(\mathbf{A}_i, \mathbf{A}_i', \perp)$) for $1 \le i \le \ell$. In the *Challenge* phase, given the ciphertext matrix $\mathbf{A}'$ of some challenge instance $\mathbf{A}$, the HNF matrix $\mathbf{H}'$ of $\mathbf{A}'$ and an unimodular matrix $\mathbf{V}'$ satisfying $\mathbf{V}'\mathbf{H}' = \mathbf{A}'$, the adversary tries to obtain the HNF of $\mathbf{A}$.

According to the client verification and decryption algorithm, $\mathbf{A}$ and $\mathbf{B} = \mathbf{H}'\mathbf{R}^{-1}$ share the same HNF. Since $Rank(\mathbf{H}') = n$, the mapping $g(\mathbf{r}) = \mathbf{H}'\mathbf{r}$ is an injective function from $\mathbb{R}^n$ to $\mathbb{R}^m$. That is, the number of different $\mathbf{B}$s is the same with that of $\mathbf{R}$s. Since the entries in $\mathbf{R}$ are chosen uniformly at random from $\mathbb{Z} \cap (-2^{\lambda}, 2^{\lambda})$, the probability that the adversary can obtain the HNF of $\mathbf{A}$ is

$$\begin{aligned}&\frac{1}{|\{\mathbf{B} \mid \mathbf{B} = \mathbf{H}'\mathbf{R}^{-1}\}|}\\ &\quad\le \frac{1}{|\{\mathbf{R} \mid \mathbf{R} \text{ is constructed as in Alg.1}\}|}\\ &\quad\le \frac{1}{(2(2^{\lambda}-1))^{n+1}(2\cdot 2^{\lambda}-1)^{n-1}},\end{aligned}$$

which also is a negligible function of n. $\square$

C. VERIFIABILITY

Theorem 3: For any large-scale input matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$ *of rank n, the proposed algorithm* $\mathrm{OAlg}_{HNF}(\mathbf{A})$ *is* 1*-verifiable according to Definition 3.*

Proof: Corresponding to our algorithm, $y' = \mathcal{F}'(x')$ in Definition 3 means

(C) $\mathbf{H}'$ is in HNF $\wedge$ $\mathbf{V}'$ is unimodular $\wedge$ $\mathbf{V}'\mathbf{H}' = \mathbf{A}'$.

According to Definition 3, we need to prove (1) the probability of the algorithm **Verify&Decrypt** outputting $\mathbf{H}$ in case that the condition (C) holds is 1, and (2) the probability of the algorithm **Verify&Decrypt** outputting $\perp$ in case that the condition (C) fails is 0. The proof of (1) directly follows the correctness of our algorithm in Theorem 1, and the proof of (2) is easily obtained from the description of **Verify&Decrypt** in Algorithm 4.

$\square$

D. EFFICIENCY

This section strictly analyzes the proposed algorithm's theoretical effectiveness. Namely, we have

Theorem 4: For any large-scale input matrix $\mathbf{A} \in \mathbb{Z}^{m\times n}$ *of rank n, the efficiency factor α achieved by our proposed algorithm* $\mathrm{OAlg}_{HNF}(\mathbf{A})$ *according to Definition 4 is*

$$\tilde{O}\left(\frac{n^{\theta}\log\|\mathbf{A}\|}{(mn+m^{\theta-1})(\lambda+\log\|\mathbf{A}\|)}\right).$$

In particular, take $\lambda = O(\log\|\mathbf{A}\|)$*, we have*

$$\alpha = \tilde{O}\left(\frac{n^{\theta}}{mn+m^{\theta-1}}\right),$$

and further, if $m = n$*, then*

$$\alpha = \tilde{O}\left(n^{\theta-2}\right),$$

where $2.3728639 \le \theta \le 3$.

Proof: By Lemma 6, without outsourcing, the client's time cost is $t_{\text{original}} = \tilde{O}(mn^{\theta-1}M(n\log\|\mathbf{A}\|, n\log\|\mathbf{A}\|))$.

In our proposed algorithm, let $t_{\mathbf{KeyGen}}$, $t_{\mathbf{Encrypt}}$, $t_{\mathbf{Verify\&Decrypt}}$ denote the client-side time costs of the sub-algorithms **KeyGen**, **Encrypt** and **Verify&Decrypt** respectively.

(1) Estimation of $t_{\mathbf{KeyGen}}$. In the key generation algorithm, we need to generate a random block upper-triangular matrix $\mathbf{R}$, a random permutation matrix $\mathbf{P}$ and a random sparse unimodular matrix $\mathbf{U}$. Clearly, generating $\mathbf{R}$ needs $O(nM(\lambda, \lambda))$ bit operations. The permutation matrix can be generated by employing the classic algorithm given by Durstenfeld [13] (Knuth [23] attributes the algorithm to Tippett [46]), which requires at most n swap operations. Also, the unimodular matrix $\mathbf{U}$ can be constructed by utilizing the well-known extended Euclidean algorithm in time $O(m\lambda M(\lambda, \lambda))$. Totally, $t_{\mathbf{KeyGen}} = O((n + m\lambda)M(\lambda, \lambda))$.

(2) Estimation of $t_{\mathbf{Encrypt}}$. In the client encryption algorithm, the ciphertext matrix $\mathbf{A}' = (\mathbf{U}(\mathbf{PA}))\mathbf{R}$ can be computed by consecutively employing matrix multiplication between a

sparse matrix and a dense matrix, and thus the time complexity is $t_{\mathbf{Encrypt}} = O(mnM(\lambda, \log\|\mathbf{A}\|) + mnM(\lambda + \log\|\mathbf{A}\|, \lambda)) = O(mnM(\lambda + \log\|\mathbf{A}\|, \lambda))$.

(3) Estimation of $t_{\mathbf{Verify\&Decrypt}}$. In the client verification and decryption algorithm, the client's time cost mainly contains five parts. Let $t_{\mathbf{H}'}$, $t_{\mathbf{V}'}$, $t_{\mathbf{R}^{-1}}$, $t_{\mathbf{B}}$ and $t_{\mathbf{H}}$ denote the time cost of checking $\mathbf{V}'\mathbf{H}' = \mathbf{A}'$, computing $\det(\mathbf{V}')$, computing $\mathbf{R}^{-1}$, computing $\mathbf{B}$ and recovering $\mathbf{H}$ from $\mathbf{B}$ respectively. According to Lemma 6, the size of entires in $\mathbf{V}'$ is bounded by $\tilde{O}(\log\|\mathbf{A}'\| + \log n)$ and the total size of $\mathbf{H}'$ is bounded by $\tilde{O}(mn\log\|\mathbf{A}'\|)$. Hence, $t_{\mathbf{H}'}$ has the order of $\tilde{O}(m^2\, nM(\log\|\mathbf{A}'\|, \log\|\mathbf{A}'\|)) = \tilde{O}(m^2\, nM(\lambda + \log\|\mathbf{A}\|, \lambda + \log\|\mathbf{A}\|))$, and the time cost of computing the determinant of $\mathbf{V}'$ is $t_{\mathbf{V}'} = \tilde{O}(m^\theta M(\log\|\mathbf{A}'\|, \log\|\mathbf{A}'\|)) = \tilde{O}(m^\theta M(\lambda + \log\|\mathbf{A}\|, \lambda + \log\|\mathbf{A}\|))$. By Lemma 3,

$$\mathbf{R}^{-1} = \begin{pmatrix} \mathbf{R}_1^{-1} & -\mathbf{R}_1^{-1}\mathbf{R}_2\mathbf{R}_3^{-1} \\ \mathbf{0} & \mathbf{R}_3^{-1} \end{pmatrix}$$

is a block triangular matrix. Since $\mathbf{R}_3$ is an $(n-2) \times (n-2)$ sparse triangular matrix, $t_{\mathbf{R}^{-1}}$ can be bound by $O(n^2M(\lambda, \lambda))$. Since the total size of $\mathbf{H}'$ is $\tilde{O}(mn\log\|\mathbf{A}'\|)$ bits and $\mathbf{B} = \mathbf{H}'\mathbf{R}^{-1}$ is block triangular, $t_{\mathbf{B}}$ is on the order of $\tilde{O}(mn^2M(\log\|\mathbf{A}'\|, \lambda)) = \tilde{O}(mn^2M(\lambda + \log\|\mathbf{A}\|, \lambda))$. By Lemma 7, the asymptotic fastest known algorithm of computing the HNF of the matrix $\mathbf{B}$ is with a time complexity of $t_{\mathbf{H}} = O(n^2M(\log D, \log D))$, where $D = \det(\mathbf{B})$. Since $D = \det(\mathbf{B}) = \det(\mathbf{A}) \leq n^{n/2}\|\mathbf{A}\|$, $t_{\mathbf{H}} = O(n^2M(n\log\|\mathbf{A}\|, n\log\|\mathbf{A}\|))$. Thus,

$$\begin{aligned} t_{\mathbf{Verify\&Decrypt}} &= t_{\mathbf{H}'} + t_{\mathbf{V}'} + t_{\mathbf{R}^{-1}} + t_{\mathbf{B}} + t_{\mathbf{H}} \\ &= \tilde{O}(m^2\, nM(\lambda + \log\|\mathbf{A}\|, \lambda + \log\|\mathbf{A}\|)) \\ &\quad + \tilde{O}(m^\theta M(\lambda + \log\|\mathbf{A}\|, \lambda + \log\|\mathbf{A}\|)) \\ &\quad + O(n^2M(\lambda, \lambda)) + \tilde{O}(mn^2M(\lambda + \log\|\mathbf{A}\|, \lambda)) \\ &\quad + O(n^2M(n\log\|\mathbf{A}\|, n\log\|\mathbf{A}\|)) \\ &= \tilde{O}\Big((m^2n + m^\theta)M(\lambda + \log\|\mathbf{A}\|, \lambda + \log\|\mathbf{A}\|) \\ &\quad + n^2M(n\log\|\mathbf{A}\|, n\log\|\mathbf{A}\|)\Big) \end{aligned}$$

To sum up, the time cost of the client in our proposed algorithm

$$\begin{aligned} t_{\text{client}} &= t_{\mathbf{KeyGen}} + t_{\mathbf{Encrypt}} + t_{\mathbf{Verify\&Decrypt}} \\ &= O((n + m\lambda)M(\lambda, \lambda)) + O(mnM(\lambda + \log\|\mathbf{A}\|, \lambda)) \\ &\quad + \tilde{O}\Big((m^2n + m^\theta)M(\lambda + \log\|\mathbf{A}\|, \lambda + \log\|\mathbf{A}\|) \\ &\quad + n^2M(n\log\|\mathbf{A}\|, n\log\|\mathbf{A}\|)\Big) \\ &= \tilde{O}\Big((m^2n + m^\theta)M(\lambda + \log\|\mathbf{A}\|, \lambda + \log\|\mathbf{A}\|) \\ &\quad + n^2M(n\log\|\mathbf{A}\|, n\log\|\mathbf{A}\|)\Big) \end{aligned}$$

Finally, noticing that $m \geq n$ and the multiplication complexity $M(x, y) = \tilde{O}(\max\{x, y\})$ as introduced in Section III-E, we have

$$\begin{aligned} t_{\text{client}} &= \tilde{O}\left((m^2n + m^\theta)(\lambda + \log\|\mathbf{A}\|) + n^3\log\|\mathbf{A}\|\right) \\ &= \tilde{O}\left((m^2n + m^\theta)(\lambda + \log\|\mathbf{A}\|)\right), \end{aligned}$$

$$\begin{aligned} t_{\text{original}} &= \tilde{O}(mn^{\theta-1}M(n\log\|\mathbf{A}\|, n\log\|\mathbf{A}\|)) \\ &= \tilde{O}(mn^\theta\log\|\mathbf{A}\|), \end{aligned}$$

and thus the efficiency factor in Definition 4 is

$$\begin{aligned} \alpha &= \frac{t_{\text{original}}}{t_{\text{client}}} \\ &= \frac{\tilde{O}(mn^\theta\log\|\mathbf{A}\|)}{\tilde{O}\left((m^2n + m^\theta)(\lambda + \log\|\mathbf{A}\|)\right)} \\ &= \tilde{O}\left(\frac{n^\theta\log\|\mathbf{A}\|}{(mn + m^{\theta-1})(\lambda + \log\|\mathbf{A}\|)}\right). \end{aligned}$$

Particularly, take $\lambda = O(\log\|\mathbf{A}\|)$, we have

$$\alpha = \tilde{O}\left(\frac{n^\theta}{mn + m^{\theta-1}}\right),$$

and further, if $m = n$, then

$$\alpha = \tilde{O}\left(n^{\theta-2}\right).$$

□

Remark 1: Based on the privacy analysis in Theorem 2 and the efficiency analysis in Theorem 4, to balance the efficiency and the security, $\lambda = O(\log\|\mathbf{A}\|)$ is alternative.

*Remark 2: Since the secret key is one-time, in practice, we can preprocess a large resource pool of random numbers and 2×2 unimodular matrices which can be randomly chosen to construct the secret matrices **R** and **U** in the key generation step. This can reduce the cost accumulation in big data processing.*

VI. PRACTICAL EXPERIMENTAL PERFORMANCE EVALUATION

Theoretical analysis shows that our design does benefit the client. We will continue to implement the proposed algorithm to evaluate its actual performance. We simulate client-side's operations on a Windows 10 machine with Intel(R) Core(TM) i5-8500T 2.10GHz CPU and 8GB RAM, and simulate cloud-side's operations on a Ubuntu 18.04 machine with Intel(R) Xeon(R) W-2133 3.60GHz CPU and 32GB RAM. All the codes are executed in the Wolfram Mathematica 10.4 software.

A. EVALUATION METHODOLOGY AND EXPERIMENT DESCRIPTIONS

We measure client's computational savings through comparing the client's time overhead of the designed algorithm with that of the algorithm without outsourcing. Concretely, taking the same notations as that in the proof of Theorem 4, we will check (1) the variance of the client-side time overhead t_{client} as the increasing of the size of input matrices and (2) the variance of the client-side speedup $\frac{t_{\text{original}}}{t_{\text{client}}}$ as the increasing of the size of input matrices. Also, we measure the cloud's time overhead t_{cloud} and the ratio $\frac{t_{\text{original}}}{t_{\text{cloud}}}$. Ideally, a well design should not increase the time required to solve the problem, thereby the ratio $\frac{t_{\text{original}}}{t_{\text{cloud}}}$ is expected to be approximately 1.

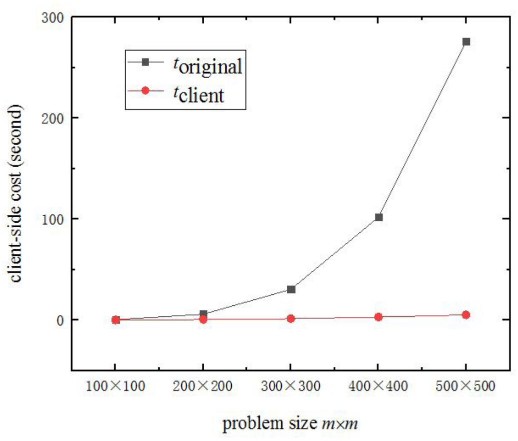


FIGURE 2. Client-side cost comparison of square matrices between algorithm OAlg$_{HNF}(\cdot)$ and algorithm without outsourcing.

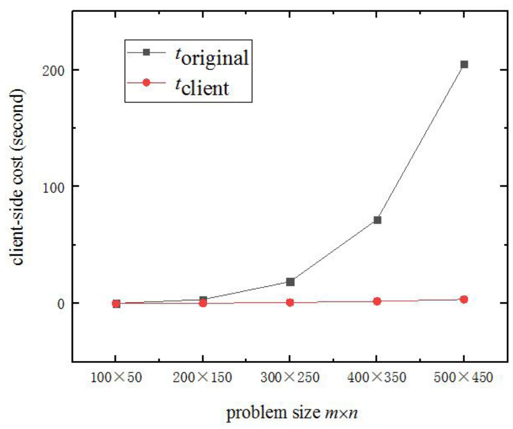


FIGURE 3. Client-side cost comparison between algorithm OAlg$_{HNF}(\cdot)$ and algorithm without outsourcing for non-square matrices.

To simulate the different circumstances in the real world, we specify the input matrices into two classes: the square integer matrices and the non-square integer matrices, whose entries are randomly chosen from $[-2^4, 2^4]$. For square integer matrices, the size ranges from 100×100 to 500×500. For non-square integer matrices, the size varies from 100×50 to 500×450. According to our theoretical analysis in Section V, in the key generation step, taking $\lambda = 4$ is appropriate for the tradeoff between the security and the efficiency.

To comprehensively evaluate the practical efficiency, we simulate all stages of our proposed algorithm. Moreover, for more detailed presentation, we divide the sub-algorithm **Verify&Decrypt** into algorithm **Verify** and algorithm **Decrypt**. Out of which, **Verify** includes checking $\mathbf{H}'$, $\mathbf{V}'$ and verifying $\mathbf{V}'\mathbf{H}' = \mathbf{A}'$, $|\det(\mathbf{V}')| = 1$, and **Decrypt** represents computing $\mathbf{B} = \mathbf{H}'\mathbf{R}^{-1}$ and its HNF matrix $\mathbf{H}$. Then, the client-side time overhead $t_{\text{client}} = t_{\textbf{KeyGen}} + t_{\textbf{Encrypt}} + t_{\textbf{Verify\&Decrypt}} = t_{\textbf{KeyGen}} + t_{\textbf{Encrypt}} + t_{\textbf{Verify}} + t_{\textbf{Decrypt}}$, and, theoretically, the client-side speedup $(t_{\text{original}}/t_{\text{client}})$ should be a considerable positive number greater than 1.

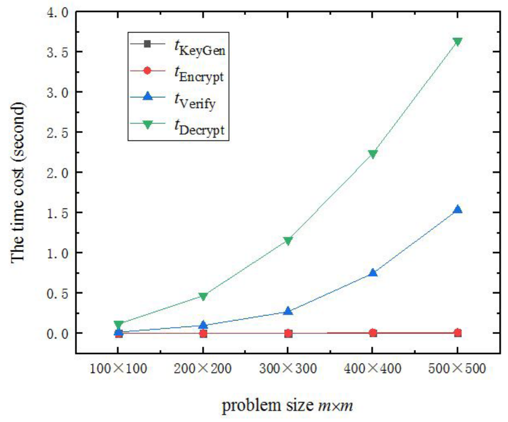


FIGURE 4. The time cost of each stage of square matrices in our proposed outsourcing algorithm OAlg$_{HNF}(\cdot)$.

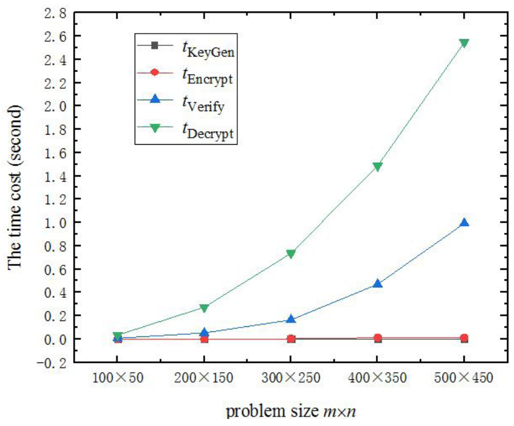


FIGURE 5. The time cost of each stage in our proposed outsourcing algorithm OAlg$_{HNF}(\cdot)$ for non-square matrices.

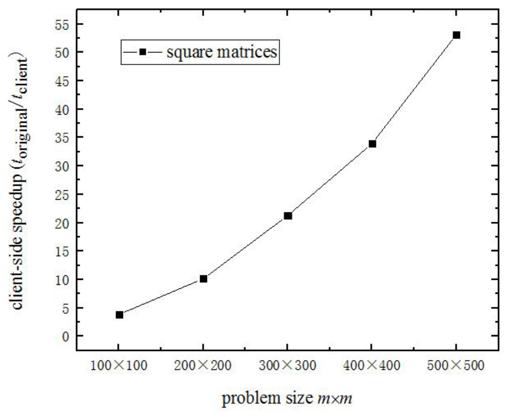


FIGURE 6. The comparison of the client-side speedup for square matrices.

B. EXPERIMENTAL RESULTS

Table 1 and Table 2 list the time cost in different stages for different sizes of square matrices and non-square matrices, respectively. FIGURE 2 - FIGURE 6 visualize the tedious data in Table 1 and Table 2. Precisely, FIGURE 2 and FIGURE 3 show a visual efficiency comparison between our proposed outsourcing algorithm and the algorithm without outsourcing. Clearly, our proposed algorithm can greatly

TABLE 1. The experimental results of square matrices.

problem size	$t_{\mathbf{KeyGen}}$	$t_{\mathbf{Encrypt}}$	$t_{\mathbf{Verify}}$	$t_{\mathbf{Decrypt}}$	t_{client}	t_{cloud}	t_{original}	$t_{\text{original}}/t_{\text{client}}$	$t_{\text{original}}/t_{\text{cloud}}$
100×100	0.000116	0.000460	0.016701	0.116500	0.133777	0.627884	0.514911	3.849025	0.820073
200×200	0.000337	0.001441	0.100027	0.470643	0.572448	7.492160	5.810810	10.150808	0.775585
300×300	0.000588	0.002603	0.270421	1.163180	1.436792	41.346800	30.534000	21.251510	0.738485
400×400	0.002717	0.010584	0.748126	2.240540	3.001967	139.379000	101.901000	33.944744	0.731107
500×500	0.004021	0.013387	1.538540	3.640830	5.196778	376.643000	276.119000	53.132730	0.733105

TABLE 2. The experimental results of non-square matrices.

problem size	$t_{\mathbf{KeyGen}}$	$t_{\mathbf{Encrypt}}$	$t_{\mathbf{Verify}}$	$t_{\mathbf{Decrypt}}$	t_{client}	t_{cloud}	t_{original}	$t_{\text{original}}/t_{\text{client}}$	$t_{\text{original}}/t_{\text{cloud}}$
100×50	0.000082	0.000343	0.007048	0.033998	0.041471	0.243453	0.178604	4.306720	0.733628
200×150	0.000335	0.001308	0.052919	0.273913	0.328475	4.125520	3.495490	10.641571	0.847285
300×250	0.000898	0.003085	0.164623	0.736614	0.905220	25.382700	18.733700	20.695190	0.738050
400×350	0.001975	0.012428	0.471040	1.486270	1.971713	98.057600	71.864500	36.447749	0.732880
500×450	0.003204	0.012550	0.993828	2.545990	3.555572	273.159000	204.943000	57.639952	0.750270

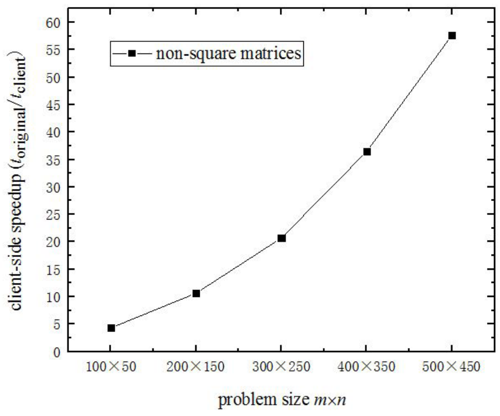


FIGURE 7. The comparison of the client-side speedup for non-square matrices.

reduce the time cost on the client-side no matter that the input matrix **A** is square or not. FIGURE 4 and FIGURE 5 compare the client-side time cost of the four stages in the proposed algorithm. It shows during the execution of the designed algorithm $\text{OAlg}_{HNF}(\cdot)$, the most time-consuming step on the client side is the process of **Verify**&**Decrypt**. Finally, FIGURE 6 and FIGURE 7 show the client-side speedup ($t_{\text{original}}/t_{\text{client}}$) for square matrices and non-square matrices with different sizes. As can be seen from the tables and figures, the designed outsourcing algorithm enables the client to achieve significant computational savings, and, with the increase of the problem sizes, the efficiency superiority of our outsourcing algorithm becomes more and more remarkable.

VII. CONCLUSION

In this paper, we present the first framework for secure and efficient HNF computation outsourcing. Our design enables a resource-constrained client to leverage the power of the cloud to securely compute the HNF of some integer matrix. Through delicate unimodular matrix and triangular matrix transformations, we manage to shift the time-consuming processing to the cloud side. Our design operates under the increasingly popular malicious single-server model and provides the first solution for securely and efficiently outsourcing the HNF computation task. We craft our design to fulfill the one-way privacy of the client's input/output information, the verifiability of the cloud's returned result. In addition, we identify our designed algorithm's efficiency both in theory and in practice, and the experimental evaluation shows the considerable computational savings on the client side.

REFERENCES

[1] M. J. Atallah and K. B. Frikken, "Securely outsourcing linear algebra computations," in *Proc. 5th ACM Symp. Inf., Comput. Commun. Secur. ASIACCS*, 2010, pp. 48–59.

[2] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. E. Spafford, "Secure outsourcing of scientific computations," in *Advances in Computers*, vol. 54. Amsterdam, The Netherlands: Elsevier, 2002, pp. 215–272.

[3] D. Benjamin and M. J. Atallah, "Private and cheating-free outsourcing of algebraic computations," in *Proc. 6th Annu. Conf. Privacy, Secur. Trust*, Oct. 2008, pp. 240–245.

[4] D. S. Bernstein, *Matrix Mathematics: Theory, Facts, and Formulas With Application to Linear Systems Theory*, vol. 41, Princeton, NJ, USA: Princeton Univ. Press, 2005.

[5] M. Bóna, *Combinatorics Permutations*. London, U.K.: Chapman & Hall, 2016.

[6] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in *Theory Cryptography*, J. Kilian, Ed. Berlin, Germany: Springer, 2005, pp. 325–341.

[7] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. S. Wong, "New algorithms for secure outsourcing of large-scale systems of linear equations," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 1, pp. 69–78, Jan. 2015.

[8] T.-W.-J. Chou and G. E. Collins, "Algorithms for the solution of systems of linear diophantine equations," *SIAM J. Comput.*, vol. 11, no. 4, pp. 687–708, Nov. 1982.

[9] J. E. Cremona and H. Cohen, "A course in computational algebraic number theory," *Math. Gazette*, vol. 78, no. 482, p. 221, Jul. 1994.

[10] Q. Ding, G. Weng, G. Zhao, and C. Hu, "Efficient and secure outsourcing of large-scale linear system of equations," *IEEE Trans. Cloud Comput.*, early access, Nov. 12, 2018, doi: 10.1109/TCC.2018.2880181.

[11] P. D. Domich, R. Kannan, and L. E. Trotter, "Hermite normal form computation using modulo determinant arithmetic," *Math. Oper. Res.*, vol. 12, no. 1, pp. 50–59, Feb. 1987.

[12] J. Duan, J. Zhou, and Y. Li, "Secure and verifiable outsourcing of large-scale nonnegative matrix factorization (NMF)," *IEEE Trans. Services Comput.*, early access, Apr. 15, 2020, doi: 10.1109/TSC.2019.2911282.

[13] R. Durstenfeld, "Algorithm 235: Random permutation," *Commun. ACM*, vol. 7, no. 7, p. 420, Jul. 1964.

[14] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *Proc. ACM Conf. Comput. Commun. Secur. CCS*, 2012, pp. 501–512.

[15] F. L. Gall, "Powers of tensors and fast matrix multiplication," in *Proc. Int. Symp. Symbolic Algebr. Comput., ISSAC*, K. Nabeshima, K. Nagasaka, F. Winkler, Á. Szántó, Eds. Kobe, Japan, Jul. 2014, pp. 296–303.

[16] C. Gentry, S. Halevi, and V. Vaikuntanathan, "A simple BGN-type cryptosystem from LWE," in *Advances in Cryptology—EUROCRYPT*, H. Gilbert, Ed. Berlin, Germany: Springer, 2010, pp. 506–522.

[17] J. L. Hafner and K. S. McCurley, ''Asymptotically fast triangularization of matrices over rings,'' *SIAM J. Comput.*, vol. 20, no. 6, pp. 1068–1083, Dec. 1991.

[18] S. Hohenberger and A. Lysyanskaya, ''How to securely outsource cryptographic computations,'' in *Proc. Theory Cryptogr. Conf.* in Lecture Notes in Computer Science, vol. 3378, 2nd ed, J. Kilian, Ed. Cambridge, MA, USA: Springer, Feb. 2005, pp. 264–282.

[19] M. S. Hung and W. O. Rom, ''An application of the Hermite normal form in integer programming,'' *Linear Algebra Appl.*, vol. 140, pp. 163–179, Oct. 1990.

[20] C. S. Iliopoulos, ''Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and smith normal forms of an integer matrix,'' *SIAM J. Comput.*, vol. 18, no. 4, pp. 658–669, Aug. 1989.

[21] R. Kannan and A. Bachem, ''Polynomial algorithms for computing the smith and Hermite normal forms of an integer matrix,'' *SIAM J. Comput.*, vol. 8, no. 4, pp. 499–507, Nov. 1979.

[22] E. Klarreich, ''Multiplication hits the speed limit,'' *Commun. ACM*, vol. 63, no. 1, pp. 11–13, Dec. 2019.

[23] D. E. Knuth, *The Art of Computer Programming, : Seminumerical Algorithms*, vol. 2, 3rd ed. Boston, MA, USA: Addison-Wesley, 1997.

[24] X. Lei, X. Liao, T. Huang, and F. Heriniaina, ''Achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud,'' *Inf. Sci.*, vol. 280, pp. 205–217, Oct. 2014.

[25] X. Lei, X. Liao, T. Huang, and H. Li, ''Cloud computing service: The caseof large matrix determinant computation,'' *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 688–700, Sep. 2015.

[26] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, ''Outsourcing large matrix inversion computation to a public cloud,'' *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, p. 1, Jan. 2013.

[27] H. Li, J. Yu, H. Zhang, M. Yang, and H. Wang, ''Privacy-preserving and distributed algorithms for modular exponentiation in IoT with edge computing assistance,'' *IEEE Internet Things J.*, early access, May 19, 2020, doi: 10.1109/JIOT.2020.2995677.

[28] D. Liu, E. Bertino, and X. Yi, ''Privacy of outsourced k-means clustering,'' in *Proc. 9th ACM Symp. Inf., Comput. Commun. Secur. ASIA CCS*, 2014, p. 123.

[29] R. Liu and Y. Pan, ''Computing Hermite normal form faster via solving system of linear equations,'' in *Proc. Int. Symp. Symbolic Algebr. Comput.*, Jul. 2019, pp. 283–290.

[30] C. Luo, K. Zhang, S. Salinas, and P. Li, ''SecFact: Secure large-scale QR and LU factorizations,'' *IEEE Trans. Big Data*, early access, Dec. 13, 2019, doi: 10.1109/TBDATA.2017.2782809.

[31] P. Meng, C. Tian, and X. Cheng, ''Publicly verifiable and efficiency/security-adjustable outsourcing scheme for solving large-scale modular system of linear equations,'' *J. Cloud Comput.*, vol. 8, no. 1, p. 24, Dec. 2019.

[32] D. Micciancio, ''Improving lattice based cryptosystems using the Hermite normal form,'' in *Cryptography Lattices*, J. H. Silverman, Ed. Berlin, Germany: Springer, 2001, pp. 126–145.

[33] D. Micciancio and B. Warinschi, ''A linear space algorithm for computing the hermite normal form,'' in *Proc. Int. Symp. Symbolic Algebraic Comput. (ISSAC)*. New York, NY, USA: Association for Computing Machinery, 2001, pp. 231–236.

[34] P. Mohassel, ''Efficient and secure delegation of linear algebra,'' Cryptol. ePrint Arch., Tech. Rep. 2011/605, 2011. [Online]. Available: https://eprint.iacr.org/2011/605

[35] M. Newman, *Integral Matrices*. New York, NY, USA: Academic, 1972.

[36] P. Paillier, ''Public-key cryptosystems based on composite degree residuosity classes,'' in *Advances in Cryptology—EUROCRYPT*, J. Stern, ed. Berlin, Germany: Springer, 1999, pp. 223–238.

[37] S. Pan, F. Zheng, W. T. Zhu, and Q. Wang, ''Harnessing the cloud for secure and efficient outsourcing of non-negative matrix factorization,'' in *Proc. IEEE Conf. Commun. Netw. Secur., CNS*, Beijing, China, May 2018, pp. 1–9.

[38] K. Ren, C. Wang, and Q. Wang, ''Security challenges for the public cloud,'' *IEEE Internet Comput.*, vol. 16, no. 1, pp. 69–73, Jan. 2012.

[39] S. Salinas, C. Luo, X. Chen, and P. Li, ''Efficient secure outsourcing of large-scale linear systems of equations,'' in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1035–1043.

[40] A. Schrijver, *Theory of Linear and Integer Programming*. Hoboken, NJ, USA: Wiley, 1998.

[41] A. Shamir, ''How to share a secret,'' *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[42] Z. Shan, K. Ren, M. Blanton, and C. Wang, ''Practical secure computation outsourcing: A survey,'' *ACM Comput. Surv.*, vol. 51, no. 2, pp. 1–40, Jun. 2018.

[43] A. Storjohann, ''Computing Hermite and smith normal forms of triangular integer matrices,'' *Linear Algebra Appl.*, vol. 282, nos. 1–3, pp. 25–45, Oct. 1998.

[44] A. Storjohann and G. Labahn, ''Asymptotically fast computation of Hermite normal forms of integer matrices,'' in *Proc. Int. Symp. Symbolic Algebr. Comput. ISSAC*, 1996, pp. 259–266.

[45] C. Tian, J. Yu, H. Zhang, H. Xue, C. Wang, and K. Ren, ''Novel secure outsourcing of modular inversion for arbitrary and variable modulus,'' *IEEE Trans. Services Comput.*, early access, Aug. 26, 2019, doi: 10.1109/TSC.2019.2937486.

[46] L. H. C. Tippett, ''Statistical tables for biological, agricultural and medical research,'' *J. Roy. Stat. Soc., Ser. C, Appl. Statist.*, vol. 2, no. 3, p. 203, 1953.

[47] V. E. Tourloupis, ''Hermite normal forms and its cryptographic applications,'' M.S. thesis, School Comput. Sci. Softw. Eng., Univ. Wollongong, Wollongong, NSW, Australia, 2013. [Online]. Available: https://ro.uow.edu.au/theses/3788

[48] C. Wang, K. Ren, J. Wang, and Q. Wang, ''Harnessing the cloud for securely outsourcing large-scale systems of linear equations,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1172–1181, Jun. 2013.

[49] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, ''Edge computing security: State of the art and challenges,'' *Proc. IEEE*, vol. 107, no. 8, pp. 1608–1631, Aug. 2019.

[50] Y. Yang, X. Huang, X. Liu, H. Cheng, J. Weng, X. Luo, and V. Chang, ''A comprehensive survey on secure outsourced computation and its applications,'' *IEEE Access*, vol. 7, pp. 159426–159465, 2019.

[51] H. Zhang, J. Yu, C. Tian, G. Xu, P. Gao, and J. Lin, ''Practical and secure outsourcing algorithms for solving quadratic congruences in Internet of Things,'' *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2968–2981, Apr. 2020.

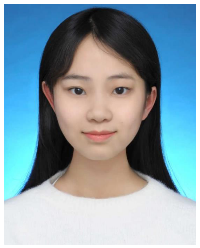
[52] L. Zhang, H. Zhang, J. Yu, and H. Xian, ''Blockchain-based two-party fair contract signing scheme,'' *Inf. Sci.*, vol. 535, pp. 142–155, Oct. 2020.

[53] S. Zhang, C. Tian, H. Zhang, J. Yu, and F. Li, ''Practical and secure outsourcing algorithms of matrix operations based on a novel matrix encryption method,'' *IEEE Access*, vol. 7, pp. 53823–53838, 2019.

[54] P. Zhao, J. Yu, H. Zhang, Z. Qin, and C. Wang, ''How to securely outsource finding the min-cut of undirected edge-weighted graphs,'' *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 315–328, 2020.

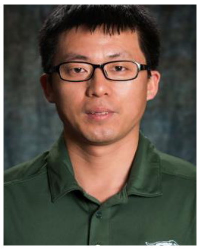
[55] L. Zhou and C. Li, ''Outsourcing eigen-decomposition and singular value decomposition of large matrix to a public cloud,'' *IEEE Access*, vol. 4, pp. 869–879, 2016.

[56] Q. Zhou, C. Tian, H. Zhang, J. Yu, and F. Li, ''How to securely outsource the extended Euclidean algorithm for large-scale polynomials over finite fields,'' *Inf. Sci.*, vol. 512, pp. 641–660, Feb. 2020.

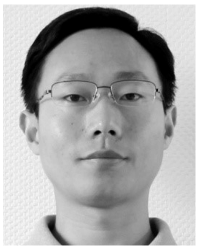


WEI ZHAO received the B.E. degree in computer science and technology from Qilu Normal University, in 2018. She is currently pursuing the M.S. degree with the College of Computer Science and Technology, Qingdao University. Her research interests include cloud computing security, secure computation outsourcing, and graph encryption.

CHENGLIANG TIAN received the B.S. and M.S. degrees in mathematics from Northwest University, Xi'an, China, in 2006 and 2009, respectively, and the Ph.D. degree in information security from Shandong University, Jinan, China, in 2013. He held a postdoctoral position with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing. He is currently with the College of Computer Science and Technology, Qingdao University, as an Associate Professor. His research interests include lattice-based cryptography and cloud computing security.



WEIZHONG TIAN received the B.S. degree in information and computing science from Northwest A&F University, China, in 2006, the M.S. degree in computational mathematics from Northwest University, Xi'an, China, in 2009, and the M.S. and Ph.D. degrees in statistics from New Mexico State University, Las Cruces, NM, USA, in 2012 and 2015, respectively. He is currently an Assistant Professor in statistics with the Department of Mathematical Sciences, Eastern New Mexico University. His research interests include family of skew slash distribution, matrix variate distribution, tail dependence, and change point detection.



YAN ZHANG (Member, IEEE) received the B.S. degree in computer science and technology from Northwestern Polytechnical University, Xi'an, China, in 2004, and the M.S. and Ph.D. degrees in automatic control from the Qingdao University of Science and Technology, Qingdao, China, in 2009 and 2014, respectively. He was a Visiting Scholar with the Research Group on Electrical Engineering and Automatic Control (GREAH), Faculty of Sciences, Normandy University, Le Havre, France, from 2014 to 2015. He is currently an Associate Professor of electrical and computer engineering with the Qingdao University of Science and Technology. His research interests include digital image processing, pattern recognition, and nondestructive testing.