IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 31, 2019, accepted April 22, 2019, date of publication April 26, 2019, date of current version May 3, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2913591

Practical and Secure Outsourcing Algorithms of Matrix Operations Based on a Novel Matrix Encryption Method

SHENGXIA ZHANG[1], CHENGLIANG TIAN[1,2,3], HANLIN ZHANG[1,2], JIA YU[1,3], AND FENGJUN LI[4]

[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 Electrical Engineering and Computer Science, The University of Kansas, Lawrence, KS 66045, USA

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

This work was supported in part by the National Natural Science Foundation of China under Grant 61702294 and Grant 61572267, in part by the Natural Science Foundation of Shandong Province under Grant ZR2016FQ02, in part by the National Development Foundation of Cryptography under Grant MMJJ20170126 and Grant MMJJ20170118, in part by the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, through the Open Research Project under Grant 2016-MS-23 and Grant 2017-MS-21, and in part by the Applied Basic Research Project of Qingdao City under Grant 17-1-1-10-jch.

ABSTRACT With the recent growth and commercialization of cloud computing, outsourcing computation has become one of the most important cloud services, which allows the resource-constrained clients to efficiently perform large-scale computation in a pay-per-use manner. Meanwhile, outsourcing large scale computing problems and computationally intensive applications to the cloud has become prevalent in the science and engineering computing community. As important fundamental operations, large-scale matrix multiplication computation (MMC), matrix inversion computation (MIC), and matrix determinant computation (MDC) have been frequently used. In this paper, we present three new algorithms to enable secure, verifiable, and efficient outsourcing of MMC, MIC, and MDC operations to a cloud that may be potentially malicious. The main idea behind our algorithms is a novel matrix encryption/decryption method utilizing consecutive and sparse unimodular matrix transformations. Compared to previous works, this versatile technique can be applied to many matrix operations while achieving a good balance between security and efficiency. First, the proposed algorithms provide robust confidentiality by concealing the local information of the entries in the input matrices. Besides, they also protect the statistic information of the original matrix. Moreover, these algorithms are highly efficient. Our theoretical analysis indicates that the proposed algorithms reduce the time overhead on the client side from $O(n^{2.3728639})$ to $O(n^2)$. Finally, the extensive experimental evaluations demonstrate the practical efficiency and effectiveness of our algorithms.

INDEX TERMS Cloud computing, outsourcing computation, matrix operations, privacy, efficiency.

I. INTRODUCTION

As one of the most important applications of cloud computing, outsourcing computation has attracted researchers' more and more attention and become a popular research topic recently [12]–[15], [42], [44]. This unprecedented and promising computing model allows clients with limited computing capability to delegate their heavy computation tasks to resource-abundant cloud servers in a pay-on-demand manner.

However, outsourcing computation also raises several security issues. First, clients' outsourced data may contain sensitive information such as business financial records, private healthcare data, or proprietary asset data, etc. that need to be carefully protected, while the cloud servers may not be fully trusted. Due to various financial incentives, the cloud server may be lazy, curious, or even malicious. As a result, it may return a random or an artificial result to the client, steal or even deliberately leak the client's private information. Moreover, the client may receive wrong computational results due to unpredictable software and hardware faults or

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

external attacks. Therefore, an ideal secure outsourcing algorithm is expected to meet the following requirements: (1) Correctness. With an honest cloud server, the algorithm should allow the client to obtain the actual results correctly. (2) Input/Output privacy. The algorithm should guarantee that the actual input and output of the computation task are appropriately hidden from the cloud sever, no matter it is honest or malicious. (3) Verifiability. The algorithm can ensure that the client can verify the correctness of the results returned from the cloud server with a nonnegligible probability. (4) Efficiency. The algorithm has to reduce the computation overhead of the client to an amount substantially smaller than that of executing the original computation task on his own.

Matrix, as a fundamental mathematical object, is widely used in many scientific and engineering computing fields. Many different problems can be phrased as matrix computation problems. In every branch of physics, including classical mechanics, electromagnetism, quantum mechanics, and quantum electrodynamics, matrices are used to study physical phenomena, such as the motion of rigid bodies [37]. In computer graphics, they are used to manipulate 3D models and project them onto a 2-dimensional screen. In probability theory and statistics, stochastic matrices are used to describe sets of probabilities, e.g., the PageRank algorithm denotes link relationships using matrices to rank the pages in a Google search [9]. Matrices are also used in economics to describe systems of economic relationships [45]. Moreover, in coding theory and cryptography, matrices over finite fields are used to represent codes [38] and encryption/decryption keys [1], [30]. Therefore, the algorithms with fundamental matrix operations, such as matrix multiplication computation (MMC), matrix inversion computation (MIC) and matrix determinant computation (MDC), have become an important component of modern scientific computing.

From the complexity perspective, the MMC, MIC and MDC operations have a same time complexity of $O(n^3)$ (for n by n square matrices) for the basic algorithm and $O(n^{2.3728639})$ for the asymptotically fastest known algorithms [26], [36]. However, with the exponential growth in the quantity of data generated in the era of big data, the matrices to be processed in real-world problems are often very large, typically with hundreds of thousands of entries [4], [10], [22]. On the other hand, the proliferation of Internet of Things (IoT) devices, such as wearable devices, smart appliances, and smart vehicles, raise increasing needs to process and analyze local, realtime data. With limited computing resources, it is difficult for IoT devices to accomplish such large-scale matrix operations. Consequently, designing secure, verifiable, and efficient outsourcing protocols for matrix operations becomes an imperative requirement for resource-constrained clients, especially in the IoT era.

A. THE PROBLEM AND RELATED WORK

In the past few years, motivated by the extensive applications of large-scale matrices in various fields, many scholars have proposed numerous protocols to outsource different matrix operations [3], [24], [39], [40], [50].

As one of the most fundamental matrix operations, matrix multiplication computation (MMC) outsourcing was first investigated by Atallah et al. in [3]. They initialized the study on outsourcing various matrix operations using the random permutation matrix disguise techniques. Benjamin and Atallah further proposed algorithms for secure MMC outsourcing based on an overly strong security assumption of two non-colluding servers [5]. Moreover, their protocols required performing expensive homomorphic encryption operations on the client side. Subsequently, by utilizing Shamir's secret sharing technique, Atallah and Frikken improved the aforementioned schemes and constructed an outsourcing protocol of MMC under the single malicious cloud server assumption [2]. However, in the encryption and decryption stages of their protocol, the client-side time cost is $O(t^2n^2)$. Meanwhile, the cloud server is required to conduct at least $4t + 2$ matrix multiplications, where t is the threshold in Shamir's secret sharing scheme. Moreover, to achieve high verifiability, additional matrix multiplications on noise matrices are needed. All these make their protocol suffer from poor efficiency.

To efficiently delegate the heavy computation task to the cloud server, Mohassel [32] initialized the design of secure outsourcing algorithm for matrix multiplication using several existing homomorphic encryption schemes, such as Paillier [35], BGN encryption [8], and the GHV scheme [23]. In theory, Mohassel's algorithm reduces the complexity of client-side time cost to $O(n^2)$, but it cannot achieve such efficiency in practice due to the time-consuming homomorphic encryption process. Recently, Lei et al. presented an efficient outsourcing algorithm of MMC with a concise encryption/decryption method and Monte Carlo verification [27]. The main idea is to encrypt/decrypt the matrix by multiplying a special sparse matrices (i.e. random permutation matrices). While the simplicity of permutation matrices yields a high efficiency, their algorithm suffers from some security issues, e.g. leaking the statistic information of zeros in the original matrix. Motivated by this observation, Fu et al. designed an improved algorithm to securely outsource MMC by introducing a new matrix encryption technique based on a matrix addition operation [21]. In particular, they disguised the original matrix by adding a random secret matrix. However, this technique requires carrying out small-scale matrix multiplication on the local-client side, which makes the algorithm less efficient. Furthermore, it can not be generalized to outsource other common matrix operations such as MIC and MDC. It is worthy noting that another thrust in MMC outsourcing research is designing publicly verifiable protocols [18], [19], [43], [46]–[49] to allow any third party to verify the correctness of the calculation results returned from the cloud server. While it is an important issue in matrix outsourcing, it is out of the scope of our study as we focus on the two-party model in this paper.

Matrix inversion computation (MIC) is another widely applicable algebraic operation. Mohassel first investigated the secure outsourcing of MIC and proposed a secure outsourcing algorithm [32]. The algorithm encrypts the original matrix through a random matrix transformation technique (i.e. multiplying the original matrix by a random secret matrix) and invokes the outsourcing algorithm of MMC as a subroutine. As mentioned earlier, the outsourcing algorithm of MMC is based on expensive homomorphic encryption (HE) schemes. Therefore, their algorithm is subject to the low efficiency and fails to be practical in applications. In order to avoid the time-consuming HE operations, Lei et al. further put forward a new protocol [29] by directly using a similar random permutation matrix transformation technique as used in [27]. However, the algorithm also suffers from the same security problem of exposing the information of the number of zeros in the original matrix.

Similarly, secure outsourcing matrix determinant computation (MDC) has been studied recently. Following their work on securely outsourcing MMC and MIC [27], [29], Lei et al. designed a secure outsourcing algorithm of MDC [28]. To protect the statistic information of zeros in the original matrix, they designed a two-step encryption algorithm. The first step is to embed the n-dimensional original matrix into an $(n+m)$-dimensional matrix obtained by adding an n-by-m zero matrix, an m-dimensional diagonal matrix and an m-by-n random matrix $\mathbf{B}$, which is the key ingredient to hide the number of zeros. In the second step, it encrypts the $(n+m)$-dimensional matrix via random permutations. Finally, the client delegates the lower-upper (LU) decomposition task to the cloud server. In other words, by skillfully converting the MDC operation into a high-dimensional matrix decomposition, the algorithm achieves the input/output privacy and verifiability. However, although the number of zeros in the original matrix is protected, the cloud server needs to perform a high-dimensional matrix decomposition, which increases the client's cost. Also, this technique is not feasible for outsourcing MMC and MIC.

B. OUR CONTRIBUTIONS

To better balance the efficiency and privacy of the outsourcing algorithm, e.g. to protect the statistic information of entries in original matrices without reducing the theoretical efficiency, this paper further explores the essence of sparse matrix transformation, and presents new secure outsourcing algorithms for MMC, MIC and MDC based on a novel matrix encryption/decryption method. This novel technique realizes the input/output privacy by encrypting a matrix in virtue of consecutive and sparse unimodular matrix transformations. The denseness of the product matrix of these sparse unimodular matrices ensures the security of the proposed algorithms. In other words, compared with previous works, this nice property guarantees that our algorithms not only conceal the location information of the entries in the original matrices, but also protect the statistical information of certain element, e.g. the number of zero in the original matrices. Meanwhile, the associativity of matrix multiplication makes our algorithms efficient. Besides, our algorithms also achieve the following properties:

1) The proposed algorithms are designed under the single untrusted program model. Our algorithms only require one cloud server and one round interaction between the client and the cloud server, which avoids the overly strong security assumption of two or more non-colluding cloud servers and achieves the optimal communication overhead.
2) The proposed algorithms have robust cheating resistance. Following the Monte Carlo verification algorithm in prior work, our algorithms allow the client to verify the correctness of the results returned from the cloud server with a probability approximated to 1.
3) Our algorithms achieve decent computational savings on the local client side. In term of complexity, the proposed algorithms reduce the local-client's computational overhead from $O(n^{2.3728639})$ to $O(n^2)$ field multiplications. Our experimental results are consistent with the theoretical analysis results.
4) The proposed matrix encryption/decryption technique can be applied in a broad range of applications. We believe that the consecutive and sparse unimodular matrix transformation technique is of independent interest. Besides the MMC, MIC and MDC operations discussed in this paper, it can be applied to outsource many other matrix operations.

C. ROAD MAP

The rest of this paper is organized as follows: we present the system model and the associated security definitions in section II, and the preliminaries in section III. Our main secure outsourcing algorithms are proposed in Section IV. In section V, we prove the correctness of the proposed algorithms and analyze their security, followed by the practical efficiency analysis and performance evaluation in section VI. Finally, we conclude our paper in Section VII.

II. SYSTEM MODEL AND SECURITY DEFINITIONS

A. SYSTEM MODEL

As shown in Fig.1, a secure outsourcing computation model generally involves two entities: a resource-constrained client and a cloud server with sufficient computational resource. The client aims to conduct some large-scale matrix operations with the assistance of the cloud server. To protect the privacy of the input data, the client first encrypts the original matrix into a large-scale ciphertext matrix using its secret key, and then sends it to the cloud server. The cloud server performs the corresponding matrix operation over the ciphertext matrix and returns the result to the client. Finally, the client verifies the correctness of the received result. If the received result is correct, the client decrypts it using the secret key to obtain the actual computational result. Otherwise, the client rejects the received result.

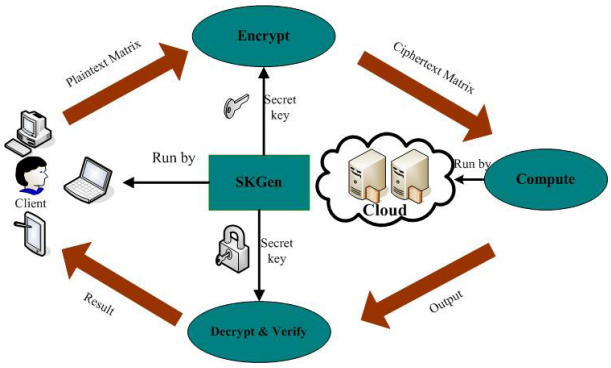


FIGURE 1. The system model.

Formally, given a computation task $\mathcal{F}$, the general framework of a secure outsourcing algorithm $\mathcal{SOA}_{\mathcal{F}} = (\textbf{SKGen}, \textbf{ClientEnc}, \textbf{CloudCom}, \textbf{ClientDec\&Ver})$ consists of the following four sub-algorithms.

1) $\textbf{SKGen}(\mathcal{F}, 1^{\kappa}) \rightarrow \{SK\}$: With the input of a computation task $\mathcal{F}$ and a security parameter κ, the secret key generating algorithm *SKGen* outputs a secret key SK, which should be kept private by the client.
2) $\textbf{ClientEnc}(\mathcal{F}, \mathbf{X}, SK) \rightarrow \{\mathbf{X}'\}$: Given a computation task $\mathcal{F}$ and the corresponding input matrix $\mathbf{X}$, the encryption algorithm *ClientEnc* uses the secret key SK to encrypt $\mathbf{X}$ into a ciphertext matrix $\mathbf{X}'$, and sends $(\mathcal{F}, \mathbf{X}')$ to the cloud server.
3) $\textbf{ClientCom}(\mathcal{F}, \mathbf{X}') \rightarrow \{\mathbf{Z}'\}$: Receiving the computation task $\mathcal{F}$ and the encrypted matrix $\mathbf{X}'$ from the client, the cloud computation algorithm *CloudCom* outputs $\mathbf{Z}' = \mathcal{F}(\mathbf{X}')$ and returns it to the client.
4) $\textbf{ClientDec\&Ver}(\mathcal{F}, \mathbf{Z}', SK) \rightarrow \{\mathbf{Z}, \perp\}$: Utilizing the secret key SK and according to different computation tasks, the algorithm *ClientDec&Ver* first decrypts the returned result $\mathbf{Z}'$ into $\mathbf{Z}$ and then verifies the correctness of $\mathbf{Z}$, or the algorithm *ClientDec&Ver* first verifies the correctness of the returned result $\mathbf{Z}'$ and then decrypts it into $\mathbf{Z}$ if $\mathbf{Z}'$ is correct. No matter in which case, if the verification succeeds, the algorithm outputs $\mathbf{Z}$, or $\perp$ otherwise.

B. THREAT MODELS

According to the behaviors of the cloud, the security threats in the outsourcing computation system can be categorized into two models: the semi-honest single-server (SS) model and the malicious single-server (MS) model.

1) SEMI-HONEST SINGLE-SERVER (SS)

The SS model, also known as the "honest but curious" single-server model, assumes the cloud server faithfully executes the delegated computation task and returns the actual result to the client. However, the cloud may be curious about the intermediate data and try to learn or infer the protected information.

2) MALICIOUS SINGLE-SERVER (MS)

In the MS model, the cloud server can arbitrarily deviate from the prescribed protocol, and attempt to learn sensitive information as much as it can. Worse still, it may intentionally return a random or forged result to the client.

Obviously, if an outsourcing algorithm is secure under the MS model, it is still secure under the SS model. As a consequence, for some given computation tasks, designing an outsoucing algorithm under the MS model, in terms of security, is more significative than that of under the SS model.

C. SYSTEM REQUIREMENTS AND SECURITY DEFINITIONS

As introduced in Section I, a well-designed secure outsourcing algorithm should at least satisfy four requirements. We reformulate their definitions as follows.

Definition 1 Correctness: Given some computation task $\mathcal{F}$, a secure outsourcing algorithm $\mathcal{SOA}_{\mathcal{F}}(\cdot)$ is correct if the secret key generating algorithm produces $SK \leftarrow \textbf{SKGen}(\mathcal{F}, 1^{\kappa})$ such that, for any valid input matrix $\mathbf{X}$, if $\mathbf{X}' \leftarrow \textbf{ClientEnc}(\mathcal{F}, \mathbf{X}, SK)$, $\mathbf{Z}' \leftarrow \textbf{CloudCom}(\mathcal{F}, \mathbf{X}')$ and $\mathbf{Z}' = \mathcal{F}(\mathbf{X}')$, then output of algorithm $\textbf{ClientDec\&Ver}(\mathcal{F}, \mathbf{Z}', SK)$ is $\mathbf{Z} = \mathcal{F}(\mathbf{X})$.

Definition 2 Input/Output privacy: Given some computation task $\mathcal{F}$, a secure outsourcing algorithm $\mathcal{SOA}_{\mathcal{F}}(\cdot)$ satisfies the input (resp. output) privacy if the secret key generating algorithm produces $SK \leftarrow \textbf{SKGen}(\mathcal{F}, 1^{\kappa})$ such that, for any valid input matrix $\mathbf{X}$, the probability that the cloud server or the adversary can recover $\mathbf{X}$ (resp. $\mathbf{Z} = \mathcal{F}(\mathbf{X})$) is negligible even if the cloud or the adversary knows the computation task $\mathcal{F}$ and $\mathbf{X}' \leftarrow \textbf{ClientEnc}(\mathcal{F}, \mathbf{X}, SK)$.

Definition 3 $(1-p)$-Verifiable: Given some computation task $\mathcal{F}$, a secure outsourcing algorithm $\mathcal{SOA}_{\mathcal{F}}(\cdot)$ is $(1-p)$-verifiable if the secret key generating algorithm produces $SK \leftarrow \textbf{SKGen}(\mathcal{F}, 1^{\kappa})$ such that, for any valid input matrix $\mathbf{X}$, if $\mathbf{X}' \leftarrow \textbf{ClientEnc}(\mathcal{F}, \mathbf{X}, SK)$, and $\mathbf{Z}' \leftarrow \textbf{CloudCom}(\mathcal{F}, \mathbf{X}')$, then the probability of $\textbf{ClientDec\&Ver}(\mathcal{F}, \mathbf{Z}', SK)$ outputting $\mathbf{Z}$ satisfies

$$\Pr[\mathbf{Z} \leftarrow \textbf{ClientDec\&Ver}(\mathcal{F}, \mathbf{Z}', SK) \mid \mathbf{Z}' \text{ is true}] = 1,$$

$$\Pr[\mathbf{Z} \leftarrow \textbf{ClientDec\&Ver}(\mathcal{F}, \mathbf{Z}', SK) \mid \mathbf{Z}' \text{ is false}] \leq p.$$

Definition 4 α-Efficient: Given some computation task $\mathcal{F}$, a secure outsourcing algorithm $\mathcal{SOA}_{\mathcal{F}}(\cdot)$ is α-efficient if, suppose the client's time overhead of performing the task on its own is t_o, and the local-client's time overhead of performing the task by employing the outsourcing algorithm $\mathcal{SOA}_{\mathcal{F}}(\cdot)$ is t_c, $\frac{t_o}{t_c} \geq \alpha$.

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. column vectors). $\mathbf{M}^{-1}$, $\mathbf{M}^{\mathrm{T}}$ and $\det(\mathbf{M})$ denote the inversion, the transpose and the determinant of the matrix $\mathbf{M}$, respectively. For convenience, some terms

frequently used in this paper, such as "matrix multiplication computation" , "matrix inversion computation" and "matrix determinant computation", are denoted by their abbreviations "MMC","MIC" and "MDC", respectively.

B. FINITE FIELD

Definition 5 Finite Field [31]: In mathematics, a finite field (i.e. Galois field) $< \mathbb{F}_q, \circ, \star, >$ *is a triple consisting of a finite set* $\mathbb{F}_q$ *with q elements, two binary operation "*$\circ$*" (addition) and "*$\star$*" (multiplication) over* $\mathbb{F}_q$ *such that*

1) $< \mathbb{F}_q, \circ >$ *is a Abelian group with addition identity* $0_{\mathbb{F}_q}$.
2) $< \mathbb{F}_q^*, \star >$ *is a Abelian group with multiplication identity* $1_{\mathbb{F}_q}$, *where* $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0_{\mathbb{F}_q}\}$.
3) *Distribution law. That is, for any* $x, y, z \in \mathbb{F}_q$, $x \star (y \circ z) = (x \star y) \circ (x \star z)$ *and* $(y \circ z) \star x = (y \star x) \circ (z \star x)$,

where q is called the order of the finite field.

A well-known property is that a finite field of order q exists if and only if the order $q = p^k$, where p is a prime number and k is a positive integer.

Example 1: Take $\mathbb{F}_q = \mathbb{F}_7$ as the finite set $\{0, 1, 2, 3, 4, 5, 6\}$, and for any $x, y \in \mathbb{F}_7$, define $x \circ y = (x + y) \mod 7$, $x \star y = (xy) \mod 7$. Then, $< \mathbb{F}_7, \circ, \star >$ is a finite field of order 7.

1) $< \mathbb{F}_7, \circ >$ is a Abelian group with addition identity 0. The additive inverses are listed as: $-0 = 0, -1 = 6, -2 = 5, -3 = 4, -4 = 3, -5 = 2, -6 = 1$.
2) $< \mathbb{F}_7^*, \star >$ is a Abelian group with multiplication identity 1, where $\mathbb{F}_7^* = \mathbb{F}_7 \setminus \{0\} = \{1, 2, 3, 4, 5, 6\}$. The multiplicative inverses are listed as: $1^{-1} = 1, 2^{-1} = 4, 3^{-1} = 5, 4^{-1} = 2, 5^{-1} = 3, 6^{-1} = 6$.
3) The multiplication is distributive over the addition.

C. PERMUTATION MAPPING AND PERMUTATION MATRIX

Other concepts used in this work are "permutation mapping" and the associated "permutation matrix". Both are widely used in group theory, combinatorics, coding theory and cryptography. So, we present their formal definitions and illustrate them with examples.

Definition 6 Permutation Mapping [7]: 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* $1, \cdots, n$.

Two formalized representations of a permutation mapping are *Cycle Notation* and *Transposition Notation*. According to the parity of the number of transpositions (two-element exchanges) contained in their representations, all the permutations can be classified as even or odd. A permutation mapping is even (resp. odd) if its representation consists of an even (resp. odd) number of transpositions. For any permutation π, the sign of π is defined as:

$$\text{sgn}(\pi) = \begin{cases} 1 & \text{if } \pi \text{ is even} \\ -1 & \text{if } \pi \text{ is odd.} \end{cases}$$

Example 2: Let $n = 3$, then

$$\begin{aligned} \pi &= \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} (\textit{Cycle Notation}) \\ &= \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 3 \end{pmatrix} (\textit{Transposition Notation}) \end{aligned}$$

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

Definition 7 Permutation Matrix [6]: 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* $1 \le i, j \le n$, *where* $\delta_{x,y}$ *is the Kronecker delta function defined as*

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

It is easy to verify that the determinant of $\mathbf{P}_\pi$ equals $\text{sgn}(\pi)$. Namely, $\det(\mathbf{P}_\pi) = \text{sgn}(\pi)$.

Example 3: Take the permutation mapping π as shown in example 2, 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, $\det(\mathbf{P}_\pi) = 1$.

D. PLU FACTORIZATION

Definition 8 PLU Factorization [16]: Let $\mathbf{A}$ *be a square matrix over a (finite) field. The* **PLU** *factorization of* $\mathbf{A}$ *refers to decomposing* $\mathbf{A}$ *into three factors – a permutation matrix* $\mathbf{P}$, *a lower triangular matrix* $\mathbf{L}$ *and an upper triangular matrix* $\mathbf{U}$, *that is* $\mathbf{A} = \mathbf{PLU}$.

Any n-by-n matrix $\mathbf{A}$ can be factorized in "**PLU**" form [34]. For example, for the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 4 \\ 1 & 4 & 5 \\ 2 & 6 & 5 \end{pmatrix} \in \mathbb{F}_7^{3\times 3},$$

its **PLU** factorization is

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 4 \\ 0 & 2 & 1 \\ 0 & 0 & 3 \end{pmatrix}.$$

E. UNIMODULAR MATRIX

Definition 9 Unimodular Matrix [41]: A matrix $\mathbf{U} \in \mathbb{F}_q^{n\times n}$ *is unimodular if and only if its determinant* $\det(\mathbf{U}) = 1_{\mathbb{F}_q}$.

A simple property of unimodular matrix is that the inverse of a unimodular matrix is also unimodular. We list it as a lemma and omit its proof.

Lemma 1 [33]: If $\mathbf{U} \in \mathbb{F}_q^{n\times n}$ *is unimodular, then there exists a unique matrix* $\mathbf{V} \in \mathbb{F}_q^{n\times n}$ *s.t.* $\det(\mathbf{V}) = 1_{\mathbb{F}_q}$ *and*

$\mathbf{UV} = \mathbf{I}_{n\times n}$, *where* $\mathbf{I}_{n\times n}$ *denotes the identity matrix over* $\mathbb{F}_q$. *Particularly, if*

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

then

$$\mathbf{V} = \begin{pmatrix} u_{22} & -u_{12} \\ -u_{21} & u_{11} \end{pmatrix} \in \mathbb{F}_q^{2\times 2}.$$

Example 4: Take $\mathbf{U} = \begin{pmatrix} 2 & 6 \\ 3 & 6 \end{pmatrix} \in \mathbb{F}_7^{2\times 2}$. Clearly, its determinant $\det(\mathbf{U}) = 2 \times 6 - 3 \times 6 = 1_{\mathbb{F}_7}$ and then $\mathbf{U}$ is unimodular. The inverse of $\mathbf{U}$ is $\mathbf{V} = \begin{pmatrix} 6 & 1 \\ 4 & 2 \end{pmatrix} \in \mathbb{F}_7^{2\times 2}$ which is obviously unimodular.

Now, we prove another result about the number of unimodular matrices over $\mathbf{F}_q$, which will be used in the security analysis of our proposed algorithms in Section V-B.

Lemma 2: Given some finite field $\mathbf{F}_q$ *and assume that the set*

$$\mathcal{L} = \left\{ \mathbf{U} \mid \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \in \mathbb{F}_q^{2\times 2} \text{ and } \det(\mathbf{U}) = 1_{\mathbf{F}_q} \right\},$$

then the size of $\mathcal{L}$ *is*

$$\#\mathcal{L} = q(q-1)(q+1).$$

Proof: Since $\det(\mathbf{U}) = 1_{\mathbf{F}_q}$, $u_{11}u_{22} - u_{21}u_{12} = 1_{\mathbf{F}_q}$. There exist two cases.

(1) If $u_{11} \neq 0_{\mathbf{F}_q}$, then for any $u_{21}, u_{12} \in \mathbf{F}_q$, $u_{22} = u_{11}^{-1}(1_{\mathbf{F}_q} + u_{21}u_{12})$ is unique determined. Hence, the number of $\mathbf{U}$ in this case is $(q-1)q^2$.

(2) If $u_{11} = 0_{\mathbf{F}_q}$, then $u_{12}u_{21} = 1_{\mathbf{F}_q}$ and $u_{22} \in \mathbf{F}_q$ could be arbitrary. Hence, the number of $\mathbf{U}$ in this case is $(q-1)q$.

In summary, the total number of unimodular matrices over $\mathbf{F}_q$ with order 2 is $(q-1)q^2 + (q-1)q = q(q-1)(q+1)$. □

IV. OUTSOURCING ALGORITHMS OF MATRIX OPERATIONS

A. COMPUTATION TASK DESCRIPTION AND BASIC IDEA

For any given finite field $\mathbb{F}_q$ and two large-scale dense matrices $\mathbf{X} \in \mathbb{F}_q^{n\times l}$, $\mathbf{Y} \in \mathbb{F}_q^{l\times m}$, where $q = p^k$ for some prime p and integer $k \geq 1$, the client intends to compute the product $\mathbf{Z} = \mathbf{XY}$, the inverse matrix $\mathbf{X}^{-1}$ (when $\mathbf{X}$ is an invertible square matrix) and the determinant $\det(\mathbf{X})$ (when $\mathbf{X}$ is a square matrix) by delegating the computation tasks to a public cloud server.

To protect the privacy of the input/ouput matrices, we should figure out an efficient matrix encryption/decryption method. A concise and nature way to encrypt $\mathbf{X}$ is multiplying $\mathbf{X}$ by two matrices $\mathbf{M}_l$ and $\mathbf{M}_r$ on both the left and right sides simultaneously. *i.e.* $\mathbf{Enc}(\mathbf{X}) = \mathbf{M}_l\mathbf{X}\mathbf{M}_r$. Obviously, to reduce the computational cost on the local client side, the matrices should be as sparse as possible. Based on this observation, Chen et al. proposed a sparse matrix encryption method that $\mathbf{M}_l$ and $\mathbf{M}_r$ are sparse and row diagonally dominant matrices [11]. However, just as the paper mentioned, this efficient encryption algorithm couldn't provide strong enough privacy. Clearly, to achieve high security, $\mathbf{M}_l$ and $\mathbf{M}_r$ should be as dense as possible. Therefore, the key is to designing appropriate matrices to make a tradeoff between the efficiency and security.

Noting that the matrix multiplication operation is associativity, these nice properties inspire us to construct the matrices $\mathbf{M}_l$ and $\mathbf{M}_r$ by the product of a series of consecutive sparse matrices. Without loss of generality, we assume $\mathbf{M}_l = \mathbf{U}_1 \cdots \mathbf{U}_{f(n)}$ and $\mathbf{M}_r = \mathbf{V}_1 \cdots \mathbf{V}_{g(l)}$, where $\mathbf{U}_i$, $\mathbf{V}_j$ are sparse matrices for $1 \leq i \leq f(n), 1 \leq j \leq g(l)$ and $f(n)$ (resp. $g(l)$) is some linear function of n (resp. l). Then the ciphertext of $\mathbf{X}$ can be efficiently computed by associativity $\mathbf{Enc}(\mathbf{X}) = \mathbf{U}_1 \cdots \mathbf{U}_{f(n)}\mathbf{X}\mathbf{V}_1 \cdots \mathbf{V}_{g(l)}$.

Now, the only problem left is to generate appropriate sparse matrices $\mathbf{U}_i$ and $\mathbf{V}_j$. Clearly, an "ideal" sparse matrix should at least satisfy the following two requirements. On one hand, it should not be too sparse. In fact, the permutation matrix is not a good choice because the product of permutation matrices is also a permutation matrix and thereby sparse. As mentioned before, it cannot protect the number of zeros in the matrix $\mathbf{X}$ and thus cannot provide strong security. On the other hand, the inverse of the sparse matrix should be efficiently computed. During the decryption and verification stages, we need to frequently compute the inverse of the sparse matrix.

Based on these considerations, we decide to use the unimodular matrix with order 2. Its inverse is also unimodular, which can be easily computed. This enlightens us to construct a high-dimensional sparse matrix by substituting certain diagonal elements of the identity matrix with some unimodular matrix of order 2. The form is shown in equation (3). In summary, we encrypt the input matrices by multiplying a series of consecutive, sparse and unimodular matrices. The decryption algorithm is the inversion procedure of encryption, which is in virtue of the same transformation.

B. OUTSOURCING ALGORITHM OF MMC

Given some finite field $\mathbb{F}_q$ with $q = p^k$ for some prime p and integer $k \geq 1$ and two large-scale dense matrices $\mathbf{X} \in \mathbb{F}_q^{n\times l}$, $\mathbf{Y} \in \mathbb{F}^{l\times m}$, the client C wants to compute the matrix $\mathbf{Z} = \mathbf{XY}$ with the assistance of the cloud server S. Our proposed secure outsourcing algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$ consists of four sub-algorithms as follows:

1) **SKGen**: The algorithm generates three random permutation mappings $\pi_1 : \{1, 2, \cdots, n\} \to \{1, 2, \cdots, n\}$, $\pi_2 : \{1, 2, \cdots, l\} \to \{1, 2, \cdots, l\}$ and $\pi_3 : \{1, 2, \cdots, m\} \to \{1, 2, \cdots, m\}$, and denotes their corresponding permutation matrices as $\mathbf{P}_1 \in \{0, 1\}^{n\times n}$, $\mathbf{P}_2 \in \{0, 1\}^{l\times l}$ and $\mathbf{P}_3 \in \{0, 1\}^{m\times m}$. Meanwhile, it randomly chooses $(n-1) + (l-1) + (m-1)$ unimodular matrices over $\mathbb{F}_q$ with order 2. That is,

$$\mathbf{U}^{(i)} = \begin{pmatrix} u_{11}^{(i)} & u_{12}^{(i)} \\ u_{21}^{(i)} & u_{22}^{(i)} \end{pmatrix}, \quad \mathbf{V}^{(j)} = \begin{pmatrix} v_{11}^{(j)} & v_{12}^{(j)} \\ v_{21}^{(j)} & v_{22}^{(j)} \end{pmatrix}$$

and

$$\mathbf{W}^{(k)} = \begin{pmatrix} w_{11}^{(k)} & w_{12}^{(k)} \\ w_{21}^{(k)} & w_{22}^{(k)} \end{pmatrix} \in \mathbb{F}_q^{2\times 2}$$

for $i = 1, \cdots, n-1$, $j = 1, 2, \cdots, l-1$, $k = 1, \cdots, m-1$. All the permutation matrices and unimodular matrices should be kept secretly by the client.

2) **ClientEnc**: This algorithm contains two steps. Input matrices $\mathbf{X} \in \mathbb{F}_q^{n\times l}$ and $\mathbf{Y} \in \mathbb{F}^{l\times m}$, Step 1 permutates the input matrices by using the secret matrices $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$. Namely, it computes $\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1}$ and $\mathbf{Y}' = \mathbf{P}_2\mathbf{Y}\mathbf{P}_3^{-1}$. Step 2 further substitutes $\mathbf{X}'$ by consecutive unimodular matrix transformations. Concretely, it computes

$$\mathbf{X}'' = \mathbf{U}_1(\cdots(\mathbf{U}_{n-1}(\mathbf{X}'\mathbf{V}_{l-1}^{-1}\cdots\mathbf{V}_1^{-1}))\cdots), \quad (1)$$

$$\mathbf{Y}'' = \mathbf{V}_1(\cdots(\mathbf{V}_{l-1}(\mathbf{Y}'\mathbf{W}_{m-1}^{-1}\cdots\mathbf{W}_1^{-1}))\cdots), \quad (2)$$

where

$$\mathbf{U}_i = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & u_{11}^{(i)} & u_{12}^{(i)} & \cdots & 0 \\ 0 & \cdots & u_{21}^{(i)} & u_{22}^{(i)} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{pmatrix} \in \mathbb{F}_q^{n\times n} \quad (3)$$

is the matrix generated by replacing the entries located in (i, i), $(i, i+1)$, $(i+1, i)$ and $(i+1, i+1)$ positions of the identity matrix $\mathbf{I}_{n\times n}$ with $\mathbf{U}^{(i)}$. Similarly, $\mathbf{V}_j \in \mathbb{F}_q^{l\times l}$ and $\mathbf{W}_k \in \mathbb{F}_q^{m\times m}$ are induced by $\mathbf{V}^{(j)}$ and $\mathbf{W}^{(k)}$ respectively by using the same constructions. Then the client C sends $\mathbf{X}''$ and $\mathbf{Y}''$ to the cloud server.

3) **CloudCom**: After receiving the encrypted matrices $\mathbf{X}''$ and $\mathbf{Y}''$, the cloud server S computes $\mathbf{Z}'' = \mathbf{X}''\mathbf{Y}''$, and then returns $\mathbf{Z}''$ to C.

4) **ClientDec&Ver**: After receiving the $\mathbf{Z}''$ from S, the client C first computes

$$\mathbf{Z}' = \mathbf{U}_{n-1}^{-1}(\cdots(\mathbf{U}_1^{-1}(\mathbf{Z}''\mathbf{W}_1\cdots\mathbf{W}_{m-1}))\cdots),$$

and

$$\mathbf{Z} = \mathbf{P}_1^{-1}\mathbf{Z}'\mathbf{P}_3.$$

Then, for a given constant λ and $i = 1$ to λ, the client randomly generates a column vector $\mathbf{r}_i \in \{0, 1\}^m$ and computes $\mathbf{e}_i = \mathbf{X} \times (\mathbf{Y} \times \mathbf{r}_i) - \mathbf{Z} \times \mathbf{r}_i$. If each vector $\mathbf{e}_i = \mathbf{0}$, the algorithm output $\mathbf{Z}$. Else, the algorithm returns $\perp$.

To make our algorithm more transparent, we explain it with a toy instance.

Example 5: Take the finite field $\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$ and suppose that

$$\mathbf{X} = \begin{pmatrix} 2 & 0 & 4 \\ 1 & 0 & 0 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 1 & 0 & 4 & 1 \\ 4 & 2 & 0 & 3 \\ 2 & 0 & 1 & 2 \end{pmatrix}.$$

The proposed algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$ works as follows: (1) The client first randomly generates three permutation mappings,

$$\pi_1 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \quad \pi_2 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix},$$
$$\pi_3 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$$

with the corresponding permutation matrices and their inverses as:

$$\mathbf{P}_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \mathbf{P}_1^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$
$$\mathbf{P}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{P}_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$
$$\mathbf{P}_3 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{P}_3^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Moreover, the client produces 6 random 2-by-2 unimodular matrices with their inverses:

$$\mathbf{U}^{(1)} = \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix}, \quad (\mathbf{U}^{(1)})^{-1} = \begin{pmatrix} 1 & 5 \\ 6 & 3 \end{pmatrix},$$
$$\mathbf{V}^{(1)} = \begin{pmatrix} 3 & 1 \\ 5 & 2 \end{pmatrix}, \quad (\mathbf{V}^{(1)})^{-1} = \begin{pmatrix} 2 & 6 \\ 2 & 3 \end{pmatrix},$$
$$\mathbf{V}^{(2)} = \begin{pmatrix} 4 & 1 \\ 3 & 1 \end{pmatrix}, \quad (\mathbf{V}^{(2)})^{-1} = \begin{pmatrix} 1 & 6 \\ 4 & 4 \end{pmatrix},$$
$$\mathbf{W}^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}, \quad (\mathbf{W}^{(1)})^{-1} = \begin{pmatrix} 2 & 6 \\ 6 & 1 \end{pmatrix},$$
$$\mathbf{W}^{(2)} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}, \quad (\mathbf{W}^{(2)})^{-1} = \begin{pmatrix} 1 & 6 \\ 6 & 2 \end{pmatrix},$$
$$\mathbf{W}^{(3)} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}, \quad (\mathbf{W}^{(3)})^{-1} = \begin{pmatrix} 1 & 0 \\ 5 & 1 \end{pmatrix}.$$

(2) The client encrypts matrices $\mathbf{X}$ and $\mathbf{Y}$. First, it computes

$$\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 4 & 0 \end{pmatrix},$$
$$\mathbf{Y}' = \mathbf{P}_2\mathbf{Y}\mathbf{P}_3^{-1} = \begin{pmatrix} 0 & 1 & 1 & 4 \\ 0 & 2 & 2 & 1 \\ 2 & 4 & 3 & 0 \end{pmatrix}.$$

Clearly, this step only confuses the local information of the entries in $\mathbf{X}'$ (resp.$\mathbf{Y}'$), which results in the exposure of statistic information of certain sensitive entries. Therefore, in order to blind the value information of each entry, the client further encrypts $\mathbf{X}'$ and $\mathbf{Y}'$ as follows:

$$\mathbf{X}'' = \mathbf{U}_1\mathbf{X}'\mathbf{V}_2^{-1}\mathbf{V}_1^{-1} = \begin{pmatrix} 2 & 3 & 6 \\ 0 & 2 & 3 \end{pmatrix},$$

$$\mathbf{Y}'' = \mathbf{V}_1\mathbf{V}_2\mathbf{Y}'\mathbf{W}_3^{-1}\mathbf{W}_2^{-1}\mathbf{W}_1^{-1} = \begin{pmatrix} 6 & 3 & 5 & 2 \\ 6 & 5 & 4 & 0 \\ 4 & 5 & 3 & 3 \end{pmatrix},$$

out of which $\mathbf{U}_1, \mathbf{V}_1, \mathbf{V}_2, \mathbf{W}_1, \mathbf{W}_2$, and $\mathbf{W}_3$ are induced by $\mathbf{U}^{(1)}, \mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}$, and $\mathbf{W}^{(3)}$ by using the construction method shown in equation (3), respectively. That is,

$$\mathbf{U_1} = \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{V_1} = \begin{pmatrix} 3 & 1 & 0 \\ 5 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$\mathbf{V_2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & 1 \\ 0 & 3 & 1 \end{pmatrix}, \quad \mathbf{W_1} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\mathbf{W_2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{W_3} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{pmatrix}.$$

(3) The cloud server computes

$$\mathbf{Z}'' = \mathbf{X}''\mathbf{Y}'' = \begin{pmatrix} 5 & 2 & 5 & 1 \\ 3 & 4 & 3 & 2 \end{pmatrix}$$

and returns it back to the client.

(4) The client decrypts it by computing

$$\mathbf{Z}' = \mathbf{U}_1^{-1}\mathbf{Z}''\mathbf{W}_1\mathbf{W}_2\mathbf{W}_3 = \begin{pmatrix} 0 & 1 & 1 & 4 \\ 0 & 3 & 3 & 5 \end{pmatrix}$$

and $\mathbf{Z} = \mathbf{P}_1^{-1}\mathbf{Z}'\mathbf{P}_3 = \begin{pmatrix} 3 & 0 & 5 & 3 \\ 1 & 0 & 4 & 1 \end{pmatrix}$. At last, it verifies the correctness of $\mathbf{Z}$.

C. OUTSOURCING ALGORITHM OF MIC

For any given finite field $\mathbb{F}_q$ with $q = p^k$ for some prime p and integer $k \geq 1$, the client C wants to compute the matrix $\mathbf{Z} = \mathbf{X}^{-1}$ for any non-singular matrix $\mathbf{X} \in \mathbb{F}_q^{n\times n}$. Our proposed secure outsourcing algorithm $\mathcal{SOA}_{MIC}(\mathbf{X})$ of MIC consists of the following four sub-algorithms:

1) **SKGen**: The algorithm generates two random permutation mappings $\pi_k : \{1, 2, \cdots, n\} \to \{1, 2, \cdots, n\}$ and denotes their corresponding permutation matrices as $\mathbf{P}_k \in \{0, 1\}^{n\times n}$ for $k = 1, 2$. Meanwhile, it randomly chooses $2(n-1)$ unimodular matrices over $\mathbb{F}_q$ with order 2. Let

$$\mathbf{U}^{(i)} = \begin{pmatrix} u_{11}^{(i)} & u_{12}^{(i)} \\ u_{21}^{(i)} & u_{22}^{(i)} \end{pmatrix}, \quad \mathbf{V}^{(j)} = \begin{pmatrix} v_{11}^{(j)} & v_{12}^{(j)} \\ v_{21}^{(j)} & v_{22}^{(j)} \end{pmatrix}$$

for $i, j = 1, \cdots, n-1$. All the permutation matrices and unimodular matrices should be kept secretly by the client.

2) **ClientEnc**: Input an invertible matrix $\mathbf{X} \in \mathbb{F}_q^{n\times n}$, this algorithm first permutates the input matrix by using the secret matrices $\mathbf{P}_k$ for $k = 1, 2$. Namely, it computes $\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2$. And then it encrypts $\mathbf{X}'$ by consecutive unimodular matrix transformations. Concretely, the ultimate ciphertext matrix is

$$\mathbf{X}'' = \mathbf{U}_1\left(\cdots\left(\mathbf{U}_{n-1}\mathbf{X}'\mathbf{V}_{n-1}\right)\cdots\right)\mathbf{V}_1$$

where $\mathbf{U}_i$, and $\mathbf{V}_j \in \mathbb{F}_q^{n\times n}$ are the matrices induced by $\mathbf{U}^{(i)}$ and $\mathbf{V}^{(j)}$ respectively by using the same constructions as shown in the equation (3). Finally, the client C sends $\mathbf{X}''$ to the cloud server.

3) **CloudCom**: After receiving the ciphertext matrix $\mathbf{X}''$, the cloud server S computes $\mathbf{Z}'' = (\mathbf{X}'')^{-1}$, and returns it to C.

4) **ClientDec&Ver**: The client C decrypts $\mathbf{Z}''$ returned from S by computing

$$\mathbf{Z}' = \mathbf{V}_{n-1}\left(\cdots\left(\mathbf{V}_1\mathbf{Z}''\mathbf{U}_1\right)\cdots\right)\mathbf{U}_{n-1},$$

and

$$\mathbf{Z} = \mathbf{P}_2\mathbf{Z}'\mathbf{P}_1,$$

Then, input a certain constant λ, and for $i = 1$ to λ, the client randomly generates a column vector $\mathbf{r}_i \in \{0, 1\}^n$ and computes $\mathbf{e}_i = \mathbf{X} \times (\mathbf{Z} \times \mathbf{r}_i) - \mathbf{I} \times \mathbf{r}_i$. If each vector $\mathbf{e}_i = \mathbf{0}$, the algorithm output $\mathbf{Z}$. Else, the algorithm returns $\perp$.

Also, let's illustrate the above algorithm with a simple example.

Example 6: Take the finite field $\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$ and suppose that $\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \in \mathbb{F}_7^{2\times 2}$. Then, the proposed algorithm $\mathcal{SOA}_{MIC}(\mathbf{X})$ goes as follows: (1) The client first generates two secret random permutation mappings

$$\pi_1 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \quad \pi_2 = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}.$$

Clearly, their associated permutation matrices are

$$\mathbf{P_1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \mathbf{P_2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Next, the client produces another two secret random unimodular matrices

$$\mathbf{U^{(1)}} = \begin{pmatrix} 1 & 0 \\ 6 & 1 \end{pmatrix}, \quad \mathbf{V^{(1)}} = \begin{pmatrix} 0 & 6 \\ 1 & 2 \end{pmatrix}.$$

(2) Using the permutation matrices $\mathbf{P}_1$ and $\mathbf{P}_2$, the client first encrypts $\mathbf{X}$ as

$$\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2 = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}.$$

Then, the client further encrypts $\mathbf{X}'$ by using unimodular matrix transformation and obtains the final ciphertext matrix

$$\mathbf{X}'' = \mathbf{U}_1\mathbf{X}'\mathbf{V}_1 = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}.$$

(3) The cloud computes

$$\mathbf{Z}'' = (\mathbf{X}'')^{-1} = \begin{pmatrix} 6 & 2 \\ 1 & 6 \end{pmatrix}$$

and returns it back to the client.

(4) The client decrypts it by computing

$$\mathbf{Z}' = \mathbf{V}_1\mathbf{Z}''\mathbf{U}_1 = \begin{pmatrix} 5 & 1 \\ 1 & 0 \end{pmatrix}$$

and

$$\mathbf{Z} = \mathbf{P}_2\mathbf{Z}'\mathbf{P}_1 = \begin{pmatrix} 1 & 5 \\ 0 & 1 \end{pmatrix}.$$

It can be easily verified that $\mathbf{Z} = \mathbf{X}^{-1}$.

D. OUTSOURCING ALGORITHM OF MDC

For any dense square matrix $\mathbf{X} \in \mathbb{F}_q^{n\times n}$, the client C intends to securely outsource the computation of $\det(\mathbf{X})$ to a cloud server S, where $\mathbb{F}_q$ with $q = p^k$ for some prime p and integer $k \geq 1$ is any given finite field. The proposed secure outsourcing algorithm $\mathcal{SOA}_{MDC}(\mathbf{X})$ is as follows:

1) **SKGen**: The algorithm generates two random permutation mappings $\pi_k : \{1, 2, \cdots, n\} \to \{1, 2, \cdots, n\}$ for $k = 1, 2$ and $2n$ random non-zero elements $\alpha_i, \beta_i \in \mathbb{F}_q$ for $i = 1, \cdots, n$. Define $\mathbf{P}_1(i, j) = \alpha_i\delta_{\pi_1(i),j}$, $\mathbf{P}_2(i, j) = \beta_i\delta_{\pi_2(i),j}$, for $1 \leq i, j \leq n$. Meanwhile, It randomly chooses $2(n-1)$ unimodular matrices over $\mathbb{F}_q$ with order 2.

$$\mathbf{U}^{(i)} = \begin{pmatrix} u_{11}^{(i)} & u_{12}^{(i)} \\ u_{21}^{(i)} & u_{22}^{(i)} \end{pmatrix}, \quad \mathbf{V}^{(i)} = \begin{pmatrix} v_{11}^{(i)} & v_{12}^{(i)} \\ v_{21}^{(i)} & v_{22}^{(i)} \end{pmatrix},$$

for $i = 1, \cdots, n-1$. All the permutation matrices and unimodular matrices should be kept secret by the client C.

2) **ClientEnc**: The client C first encrypts the matrix $\mathbf{X}$ by the following two steps:

$$\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2 \tag{4}$$

$$\mathbf{X}'' = \mathbf{U}_1\left(\cdots\left(\mathbf{U}_{n-1}\mathbf{X}'\mathbf{V}_{n-1}\right)\cdots\right)\mathbf{V}_1 \tag{5}$$

where $\mathbf{U}_i$,$\mathbf{V}_i \in \mathbb{F}_q^{n\times n}$ are induced by $\mathbf{U}^{(i)}$, $\mathbf{V}^{(i)}$ respectively by using the same construction as shown in the equation (3), and then sends $\mathbf{X}''$ to S.

3) **CloudCom**: After receiving the encrypted matrices $\mathbf{X}''$, the cloud server S performs **PLU** factorization such that $\mathbf{X}'' = \mathbf{PLU}$, and returns $\mathbf{P}$, $\mathbf{L}$ and $\mathbf{U}$ back to C.

4) **ClientDec&Ver**: Once receiving the matrices $\mathbf{P}$, $\mathbf{L}$ and $\mathbf{U}$, the client C first checks whether $\mathbf{P}$, $\mathbf{L}$ and $\mathbf{U}$ are a permutation matrix, a lower triangular matrix and an upper triangular matrix respectively. If they do, the client C inputs a certain constant λ, and, for $i = 1$ to λ, randomly generates a column vector $\mathbf{r}_i \in \{0, 1\}^n$ to compute $\mathbf{e}_i = \mathbf{P} \times (\mathbf{L} \times (\mathbf{U} \times \mathbf{r}_i)) - \mathbf{X} \times \mathbf{r}_i$. If each vector $\mathbf{e}_i = \mathbf{0}$, the algorithm outputs

$$\det(\mathbf{X}) = \frac{\det(\mathbf{P})(\prod_{i=1}^n \ell_{ii}u_{ii})}{\mathrm{sgn}(\pi_1)\prod_{i=1}^n \alpha_i \times \mathrm{sgn}(\pi_2)\prod_{i=1}^n \beta_i},$$

where ℓ_{ii} (resp. u_{ii}) denotes the elements in the diagonal of the matrix $\mathbf{L}$ (resp. $\mathbf{U}$). Else, the algorithm returns $\perp$.

Example 7: Take the finite field $\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$ and suppose that

$$\mathbf{X} = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 1 & 0 \\ 1 & 0 & 4 \end{pmatrix} \in \mathbb{F}_7^{3\times 3}.$$

Then, the proposed algorithm $\mathcal{SOA}_{MDC}(\mathbf{X})$ works as follows: (1) The client generates 4 secret random permutation mappings

$$\pi_1 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}, \quad \pi_2 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix},$$

and 12 random numbers $\alpha_1 = 2, \alpha_2 = 3, \alpha_3 = 1$, $\beta_1 = 1, \beta_2 = 2, \beta_3 = 1$. Then

$$\mathbf{P_1} = \begin{pmatrix} 0 & 2 & 0 \\ 3 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{P_2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 1 & 0 \end{pmatrix}.$$

Next, the client produces 4 secret 2-by-2 unimodular matrices

$$\mathbf{U^{(1)}} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}, \quad \mathbf{U^{(2)}} = \begin{pmatrix} 1 & 6 \\ 1 & 0 \end{pmatrix},$$

$$\mathbf{V^{(1)}} = \begin{pmatrix} 1 & 6 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{V^{(2)}} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix},$$

(2) The client encrypts $\mathbf{X}$ by computing

$$\mathbf{X}'' = (\mathbf{U}_1(\mathbf{U}_2(\mathbf{P}_1\mathbf{X}\mathbf{P}_2)\mathbf{V}_2)\mathbf{V}_1) = \begin{pmatrix} 2 & 2 & 4 \\ 2 & 3 & 0 \\ 6 & 2 & 6 \end{pmatrix}.$$

and sends it to the cloud server.

(3) The cloud server performs **PLU** factorization on $\mathbf{X}''$ and obtains

$$\mathbf{X}'' = \mathbf{PLU} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 2 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & 6 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 3 & 1 \end{pmatrix}.$$

Then, it returns them back to the client.

(4) The client checks the correctness of $\mathbf{P}$,$\mathbf{L}$ and $\mathbf{U}$, and decrypts the results by computing

$$\det(\mathbf{X}) = \frac{\det(\mathbf{P})(\prod_{i=1}^n \ell_{ii}u_{ii})}{\mathrm{sgn}(\pi_1)\prod_{i=1}^n \alpha_i \times \mathrm{sgn}(\pi_2)\prod_{i=1}^n \beta_i} = \frac{12}{(-6)\times(-2)} = 1.$$

V. CORRECTNESS AND SECURITY ANALYSIS

In this section, we first give strictly proofs on the correctness of the proposed algorithms, and then analyze their security according to the definitions modeled in section II-C.

A. CORRECTNESS

Informally, the correctness means that the outsourcing algorithms can make the client obtain the actual results correctly if the cloud server perform the delegated computation task honestly. The strict theoretical proof is given in the following theorem.

Theorem 1: Given some fixed finite field $\mathbb{F}_q$, for any valid (large-scale and dense) input matrix $\mathbf{X} \in \mathbb{F}_q^{n\times l}$, $\mathbf{Y} \in \mathbb{F}_q^{l\times m}$, the proposed algorithms $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$, $\mathcal{SOA}_{MIC}(\mathbf{X})$ and $\mathcal{SOA}_{MDC}(\mathbf{X})$ are correct according to Definition 1.

Proof: (1) The algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$ is correct. If the cloud server performs the protocol honestly,

$\mathbf{Z}'' = \mathbf{X}''\mathbf{Y}''$. Substituting $\mathbf{X}''$ and $\mathbf{Y}''$ with equations (1) and (2) respectively, we have

$$\begin{aligned}\mathbf{Z}'' &= \mathbf{X}''\mathbf{Y}'' \\ &= \left(\mathbf{U}_1 \cdots \mathbf{U}_{n-1}\mathbf{X}'\mathbf{V}_{l-1}^{-1} \cdots \mathbf{V}_1^{-1}\right) \\ &\quad \cdot \left(\mathbf{V}_1 \cdots \mathbf{V}_{l-1}\mathbf{Y}'\mathbf{W}_{m-1}^{-1} \cdots \mathbf{W}_1^{-1}\right) \\ &= \mathbf{U}_1 \cdots \mathbf{U}_{n-1}\mathbf{X}'\mathbf{Y}'\mathbf{W}_{m-1}^{-1} \cdots \mathbf{W}_1^{-1} \\ &= \mathbf{U}_1 \cdots \mathbf{U}_{n-1}(\mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1})(\mathbf{P}_2\mathbf{Y}\mathbf{P}_3^{-1})\mathbf{W}_{m-1}^{-1} \cdots \mathbf{W}_1^{-1} \\ &= \mathbf{U}_1 \cdots \mathbf{U}_{n-1}\mathbf{P}_1\mathbf{X}\mathbf{Y}\mathbf{P}_3^{-1}\mathbf{W}_{m-1}^{-1} \cdots \mathbf{W}_1^{-1}\end{aligned}$$

Thus, in the sub-algorithm **ClientDec&Ver**,

$$\mathbf{Z}' = \mathbf{U}_{n-1}^{-1} \cdots \mathbf{U}_1^{-1}\mathbf{Z}''\mathbf{W}_1 \cdot \cdots \cdot \mathbf{W}_{m-1} = \mathbf{P}_1\mathbf{X}\mathbf{Y}\mathbf{P}_3^{-1}$$

and $\mathbf{Z} = \mathbf{P}_1^{-1}\mathbf{Z}'\mathbf{P}_3 = \mathbf{X}\mathbf{Y}$.

Consequently, for any $\mathbf{r}_i \in \{0,1\}^m$, $\mathbf{e}_i = \mathbf{X} \times (\mathbf{Y} \times \mathbf{r}_i) - \mathbf{Z} \times \mathbf{r}_i = \mathbf{0}$, and thereby the algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$ returns the correct result $\mathbf{Z}$.

(2) The algorithm $\mathcal{SOA}_{MIC}(\mathbf{X})$ is correct. Similarly, if the cloud server performs the protocol honestly,

$$\begin{aligned}\mathbf{Z}'' &= (\mathbf{X}'')^{-1} = \mathbf{V}_1^{-1} \cdots \mathbf{V}_{n-1}^{-1}(\mathbf{X}')^{-1}\mathbf{U}_{n-1}^{-1} \cdots \mathbf{U}_1^{-1} \\ &= \mathbf{V}_1^{-1} \cdots \mathbf{V}_{n-1}^{-1}\mathbf{P}_2^{-1}\mathbf{X}^{-1}\mathbf{P}_1^{-1}\mathbf{U}_{n-1}^{-1} \cdots \mathbf{U}_1^{-1}.\end{aligned}$$

Thus, in the sub-algorithm **ClientDec&Ver**,

$$\mathbf{Z}' = \mathbf{V}_{n-1} \cdots \mathbf{V}_1\mathbf{Z}''\mathbf{U}_1 \cdot \cdots \cdot \mathbf{U}_{n-1} = \mathbf{P}_2^{-1}\mathbf{X}^{-1}\mathbf{P}_1^{-1}$$

and $\mathbf{Z} = \mathbf{P}_2\mathbf{Z}'\mathbf{P}_1 = \mathbf{X}^{-1}$. Consequently, for any $\mathbf{r}_i \in \{0,1\}^n$, $\mathbf{e}_i = \mathbf{X} \times (\mathbf{Z} \times \mathbf{r}_i) - \mathbf{I} \times \mathbf{r}_i = \mathbf{0}$, and thereby the algorithm $\mathcal{SOA}_{MIC}(\mathbf{X}, \mathbf{Y})$ returns the correct result $\mathbf{Z}$.

(3) The algorithm $\mathcal{SOA}_{MDC}(\mathbf{X})$ is correct. If the cloud server performs the protocol honestly,

$$\mathbf{X}'' = \mathbf{PLU}.$$

This indicates that, (i) for any $\mathbf{r}_i \in \{0,1\}^n$, $\mathbf{e}_i = \mathbf{P} \times (\mathbf{L} \times (\mathbf{U} \times \mathbf{r}_i)) - \mathbf{X} \times \mathbf{r}_i = 0$ and (ii)

$$\det(\mathbf{X}'') = \det(\mathbf{P})\det(\mathbf{L})\det(\mathbf{U}). \tag{6}$$

Meanwhile, $\mathbf{X}'' = \mathbf{U}_1 \cdots \mathbf{U}_{n-1}\mathbf{P}_1\mathbf{X}\mathbf{P}_2\mathbf{V}_{n-1} \cdots \mathbf{V}_1$ implies

$$\begin{aligned}\det\left(\mathbf{X}''\right) &= \det\left(\mathbf{U}_1 \cdots \mathbf{U}_{n-1}\mathbf{P}_1\mathbf{X}\mathbf{P}_2\mathbf{V}_{n-1} \cdots \mathbf{V}_1\right) \\ &= \det\left(\mathbf{U}_1\right) \cdots \det\left(\mathbf{U}_{n-1}\right)\det\left(\mathbf{P}_1\right) \cdot \\ &\quad \cdot \det\left(\mathbf{X}\right)\det\left(\mathbf{P}_2\right)\det\left(\mathbf{V}_{n-1}\right) \cdots \det(\mathbf{V}_1) \\ &= \det\left(\mathbf{P}_1\right)\det\left(\mathbf{X}\right)\det\left(\mathbf{P}_2\right). \end{aligned} \tag{7}$$

Combining equation (6) with equation (7), we have

$$\begin{aligned}\det(\mathbf{X}) &= \frac{\det(\mathbf{X}'')}{\det\left(\mathbf{P}_1\right)\det\left(\mathbf{P}_2\right)} = \frac{\det(\mathbf{P})\det(\mathbf{L})\det(\mathbf{U})}{\det\left(\mathbf{P}_1\right)\det\left(\mathbf{P}_2\right)} \\ &= \frac{\det(\mathbf{P}) \times \left(\prod_{i=1}^{n} \ell_{ii}u_{ii}\right)}{\mathrm{sgn}(\pi_1)\prod_{i=1}^{n}\alpha_i \times \mathrm{sgn}(\pi_2)\prod_{i=1}^{n}\beta_i}.\end{aligned}$$

which is exactly the output of the sub-algorithm **ClientDec&Ver**. Consequently, the algorithm $\mathcal{SOA}_{MDC}$ $(\mathbf{X}, \mathbf{Y})$ is correct. □

B. INPUT/OUTPUT PRIVACY

In this section, we will prove that the probability that a malicious cloud server can learn the actual input and output is negligible. Formally,

Theorem 2: Given some fixed finite field $\mathbb{F}_q$, for any valid (large-scale and dense) input matrix $\mathbf{X} \in \mathbb{F}_q^{n\times l}$, $\mathbf{Y} \in \mathbb{F}_q^{l\times m}$, the proposed algorithms $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$, $\mathcal{SOA}_{MIC}(\mathbf{X})$ and $\mathcal{SOA}_{MDC}(\mathbf{X})$ fulfill input/output privacy according to Definition 2.

Proof: (1) The algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$ fulfills input/output privacy. In terms of the input privacy, we need to prove the probability that an adversary can acquire the knowledge of $\mathbf{X}$ and $\mathbf{Y}$ is negligible even if the adversary obtains the ciphertext matrices $\mathbf{X}''$ and $\mathbf{Y}''$ and knows the encryptions algorithm **ClientEnc**. In fact, by the encryption step, we know

$$\begin{aligned}\mathbf{X}'' &= \mathbf{U}_1 \cdots \mathbf{U}_{n-1}\mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1}\mathbf{V}_{l-1}^{-1} \cdots \mathbf{V}_1^{-1}, \\ \mathbf{Y}'' &= \mathbf{V}_1 \cdots \mathbf{V}_{l-1}\mathbf{P}_2\mathbf{Y}\mathbf{P}_3^{-1}\mathbf{Y}'\mathbf{W}_{m-1}^{-1} \cdots \mathbf{W}_1^{-1}.\end{aligned}$$

By the key generation step, $\mathbf{P}_1$ is randomly chosen from a set with size $n!$, $\mathbf{P}_2$ is randomly chosen from a set with size $l!$, $\mathbf{P}_3$ is randomly chosen from a set with size $m!$, and $\mathbf{U}_i$, $\mathbf{V}_j$, $\mathbf{W}_k$ are randomly chosen from a set with size $q(q-1)(q+1)$ (see Lemma 2), where $1 \le i \le n-1$, $1 \le j \le l-1$, $1 \le k \le m-1$. Therefore, the probability that an adversary can recover $\mathbf{X}$ (resp. $\mathbf{Y}$) is

$$\frac{1}{(q(q-1)(q+1))^{n+l-2}n!l!} \left(\text{resp. } \frac{1}{(q(q-1)(q+1))^{l+m-2}m!l!}\right),$$

which is a negligible function of n (resp. m).

In terms of the output privacy, we need to prove the probability that an adversary can acquire the knowledge of $\mathbf{Z} = \mathbf{XY}$ is negligible even if the adversary obtains the ciphertext matrices $\mathbf{X}''$ and $\mathbf{Y}''$. In order to recover $\mathbf{Z}$, the adversary has two possible ways. One is to recover $\mathbf{X}$ and $\mathbf{Y}$, and then compute $\mathbf{Z} = \mathbf{XY}$. In this way, the probability is

$$\begin{aligned}&\frac{1}{(q(q-1)(q+1))^{n+l-2}n!l!} \times \frac{1}{(q(q-1)(q+1))^{l+m-2}m!l!} \\ &= \frac{1}{(q(q-1)(q+1))^{m+n+2l-4}n!m!(l!)^2},\end{aligned}$$

which obviously is a negligible function of n (resp. m). Another way is to recover $\mathbf{Z}$ from $\mathbf{Z}''$. Since

$$\mathbf{Z}'' = \mathbf{X}''\mathbf{Y}'' = \mathbf{U}_1 \cdots \mathbf{U}_{n-1}\mathbf{P}_1\mathbf{Z}\mathbf{P}_3^{-1}\mathbf{W}_{m-1}^{-1} \cdots \mathbf{W}_1^{-1},$$

and $\mathbf{P}_1$ is randomly chosen from a set with size $n!$, $\mathbf{P}_3$ is randomly chosen from a set with size $m!$, $\mathbf{U}_i$, $\mathbf{W}_k$ are randomly chosen from a set with size $q(q-1)(q+1)$ (see Lemma 2), where $1 \le i \le n-1$, $1 \le k \le m-1$, it is easily to deduce that, in this way, the probability is

$$\frac{1}{(q(q-1)(q+1))^{n+m-2}n!m!}$$

which is also a negligible function of n (resp. m).

(2) The algorithm $\mathcal{SOA}_{MIC}(\mathbf{X})$ fulfills input/output privacy. Similar to the analysis of algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$, we can prove the probability of an adversary can recover the input matrix $\mathbf{X} \in \mathbf{F}_q^{n\times n}$ is

$$\frac{1}{(q(q-1)(q+1))^{2n-2}(n!)^2},$$

which is also the probability of an adversary can recover the output matrix $\mathbf{X}^{-1}$. Obviously, it is a negligible function of n.

(3) The algorithm $\mathcal{SOA}_{MDC}(\mathbf{X})$ fulfills input/output privacy. For any valid input (large-scale and dense) matrix $\mathbf{X} \in \mathbb{F}_q^{n\times n}$, by the encryption algorithm **ClientEnc**,

$$\mathbf{X}'' = \mathbf{U}_1 \cdots \mathbf{U}_{n-1}\mathbf{P}_1\mathbf{X}\mathbf{P}_2\mathbf{V}_{n-1} \cdots \mathbf{V}_1$$

Since π_1 and π_2 are randomly chosen from a set with size $n!$, α_i and β_i are randomly chosen from $\mathbb{F}_q \backslash \{0_{\mathbb{F}_q}\}$ for $1 \le i \le n$ and $\mathbf{U}_i$ and $\mathbf{V}_i$ are randomly chosen from a set with size $q(q-1)(q+1)$ for $1 \le i \le n-1$ (see Lemma 2). It is easy to see that the probability that an adversary can recover $\mathbf{X}$ from $\mathbf{X}''$ is

$$\frac{1}{(q(q-1)(q+1))^{2n-2}(n!)^2(q-1)^{2n}}$$

which is obviously a negligible function of n.

Since

$$\begin{aligned}\det(\mathbf{X}) &= \frac{\det(\mathbf{X}'')}{\det(\mathbf{P}_1)\det(\mathbf{P}_2)} \\ &= \frac{\det(\mathbf{X}'')}{\mathrm{sgn}(\pi_1)\prod_{i=1}^{n}\alpha_i \times \mathrm{sgn}(\pi_2)\prod_{i=1}^{n}\beta_i}.\end{aligned}$$

and $\det(\mathbf{X}'')$ is public, the adversary can recover $\det(\mathbf{X})$ is by guessing the product $\det(\mathbf{P}_1)\det(\mathbf{P}_2)$ and thereby the probability is $(\frac{1}{2})^2(\frac{1}{(q-1)^n})^2$ which is obviously a negligible function of n. Also, the adversary can directly guess the value of $\det(\mathbf{X})$ by the exhaustive attack, and therein the probability is $\frac{1}{q}$. □

C. VERIFIABILITY

Informally, we need to prove the probability that a malicious server is able to fool the client with an incorrect result is negligible. Precisely, the strict result is given as follows.

Theorem 3: Given some fixed finite field $\mathbb{F}_q$, for any valid (large-scale and dense) input matrix $\mathbf{X} \in \mathbb{F}_q^{n\times l}$, $\mathbf{Y} \in \mathbb{F}_q^{l\times m}$, the proposed algorithms $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$, $\mathcal{SOA}_{MIC}(\mathbf{X})$ and $\mathcal{SOA}_{MDC}(\mathbf{X})$ are $\left(1 - \frac{1}{2^\lambda}\right)$-verifiable according to Definition 3.

Proof: We only prove the verifiability of algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$. The proofs that the algorithms $\mathcal{SOA}_{MIC}(\mathbf{X})$ and $\mathcal{SOA}_{MDC}(\mathbf{X})$ are $\left(1 - \frac{1}{2^\lambda}\right)$-verifiable are almost the same. For simplicity, we omit them.

According to Definition 3, we need to prove

$$\Pr[\mathbf{Z} \leftarrow \textbf{ClientDec\&Ver}(\mathcal{F}, \mathbf{Z}'', SK) \mid \mathbf{Z}'' \text{ is true}] = 1, \quad (8)$$

$$\Pr[\mathbf{Z} \leftarrow \textbf{ClientDec\&Ver}(\mathcal{F}, \mathbf{Z}'', SK) \mid \mathbf{Z}'' \text{ is false}] \le \frac{1}{2^\lambda}. \quad (9)$$

Following the correctness proof in Theorem 1, we immediately obtain the equation (8).

Now we analyze the inequality (9). In the sub-algorithm **ClientDec&Ver**$(\mathcal{F}, \mathbf{Z}'', SK)$, for any $1 \le i \le \lambda$, let $\mathbf{D} = \mathbf{X} \times \mathbf{Y} - \mathbf{Z}$, $\mathbf{r}_i = (r_1^{(i)}, \cdots, r_m^{(i)})^{\mathrm{T}}$ and $\mathbf{e}_i = \mathbf{X} \times (\mathbf{Y} \times \mathbf{r}_i) - \mathbf{Z} \times \mathbf{r}_i = \mathbf{D} \times \mathbf{r}_i = (e_1^{(i)}, ..., e_n^{(i)})^{\mathrm{T}}$. If $\mathbf{Z}''$ is a false result, then $\mathbf{Z}'' \ne \mathbf{X}''\mathbf{Y}''$ which further results in $\mathbf{Z} \ne \mathbf{XY}$. In other words, $\mathbf{D} \ne 0$. Hence, there exists at least one non-zero element in $\mathbf{D}$. Suppose that the element $d_{st} \ne 0$ for some $1 \le s \le n$ and $1 \le t \le m$, we obtain

$$\begin{aligned}e_s^{(i)} &= \sum_{k=1}^{m} d_{sk}r_k^{(i)} = d_{s1}r_1^{(i)} + \cdots + d_{st}r_t^{(i)} + \cdots + d_{sm}r_m^{(i)} \\ &= d_{st}r_t^{(i)} + y \end{aligned} \quad (10)$$

where $y = \sum_{k=1}^{m} d_{sk}r_k^{(i)} - d_{st}r_t^{(i)}$. According to the Total Probability Formula,

$$\begin{aligned}\Pr[e_s^{(i)} = 0] = \Pr[e_s^{(i)} = 0 | y = 0]\Pr[y = 0] \\ + \Pr[e_s^{(i)} = 0 | y \ne 0]\Pr[y \ne 0].\end{aligned} \quad (11)$$

Note that, from equation (10), we have

$$\begin{cases}\Pr[e_s^{(i)} = 0 | y = 0] = \Pr[r_t^{(i)} = 0] = 1/2, \\ \Pr[e_s^{(i)} = 0 | y \ne 0] \le \Pr[r_t^{(i)} = 1] = 1/2.\end{cases} \quad (12)$$

Substituting (12) into (11), we have

$$\Pr[e_s^{(i)} = 0] \le \frac{1}{2}\Pr[y = 0] + \frac{1}{2}\Pr[y \ne 0] = \frac{1}{2}. \quad (13)$$

Therefore,

$$\Pr[\mathbf{e}_i = (0, \cdots, 0)^T] \le \Pr[e_s^{(i)} = 0] \le 1/2.$$

Since the verification repeats λ times, we have

$$\begin{aligned}&\Pr[\mathbf{Z} \leftarrow \textbf{ClientDec\&Ver}(\mathcal{F}, \mathbf{Z}'', SK) \mid \mathbf{Z}'' \text{ is false}] \\ &\le (\Pr[\mathbf{e}_i = (0, \cdots, 0)^{\mathrm{T}}])^\lambda \le \frac{1}{2^\lambda}.\end{aligned}$$

□

VI. EFFICIENCY ANALYSIS AND PERFORMANCE EVALUATION

In this section, we provide an elaborate theoretical and experimental analysis on the efficiency of the proposed algorithms.

A. THEORETICAL ANALYSIS

Theorem 4: Given some fixed finite field $\mathbb{F}_q$, for any valid (large-scale and dense) input matrix $\mathbf{X} \in \mathbb{F}_q^{n\times l}$, $\mathbf{Y} \in \mathbb{F}_q^{l\times m}$, the proposed algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$ is $O(\frac{mln}{mn+nl+ml})$-efficient, and for square input matrices, all the proposed algorithms $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$, $\mathcal{SOA}_{MIC}(\mathbf{X})$ and $\mathcal{SOA}_{MDC}(\mathbf{X})$ are $O(n^{0.3728639})$-efficient according to Definition 4.

Proof: Without outsourcing, the client has to perform the MMC, MIC and MDC on its own. For any n-by-n square matrix, the most efficient known algorithm for these operations is introduced by François Le Gall with an asymptotic

complexity of $O(n^{2.3728639})$ [26]. For general non-square input matrices arising in MMC, the client can compute the product of $\mathbf{X} \in \mathbb{F}_q^{n\times l}$ and $\mathbf{Y} \in \mathbb{F}_q^{l\times m}$ by the conventional iterative algorithm with a complexity of $O(nlm)$.

Now, we analyze the client-side time cost in our outsourcing algorithms. In the sub-algorithm **SKGen** of the proposed algorithms, we need to generate random permutations and unimodular matrices. Utilizing the classic algorithm given by Durstenfeld [17] (Knuth [25] attributes the algorithm to Fisher and Yates [20]), it only requires at most n swap operations to generate a random permutation $\pi : \{1, \cdots, n\} \leftarrow \{1, \cdots, n\}$. Also, the generation of 2-by-2 unimodular matrices is independent of the input matrices, and thereby they are can be efficiently preprocessed by using the well-known extended Euclidean algorithm. Therefore, the principal term of the client-side time overhead consists of the cost of sub-algorithms **ClientEnc** and **ClientDec&Ver**. Let "M" denote the multiplication operation in the finite field $\mathbb{F}_q$ and we omit the cost of field additions since it is negligible compared with that of field multiplications.

(1) Client-side cost of algorithm $\mathcal{SOA}_{MMC}(\mathbf{X}, \mathbf{Y})$.

- Cost of **ClientEnc**. For any valid input matrices $\mathbf{X} \in \mathbb{F}_q^{n\times l}$ and $\mathbf{Y} \in \mathbb{F}_q^{l\times m}$, since matrix multiplication operation satisfies associative law and computing the product of a dense matrix and a sparse unimodular matrix is efficient, computing $\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1}$ and
$$\mathbf{X}'' = \mathbf{U}_1(\cdots(\mathbf{U}_{n-1}(\mathbf{X}'\mathbf{V}_{l-1}^{-1}\cdots\mathbf{V}_1^{-1}))\cdots)$$
needs to perform at most $(4n(l-1)+ 4l(n-1))$M and computing $\mathbf{Y}' = \mathbf{P}_2\mathbf{Y}\mathbf{P}_3^{-1}$ and
$$\mathbf{Y}'' = \mathbf{V}_1(\cdots(\mathbf{V}_{l-1}(\mathbf{Y}'\mathbf{W}_{m-1}^{-1}\cdots\mathbf{W}_1^{-1}))\cdots)$$
needs to perform at most $(4l(m-1) + 4m(l-1))$M. Hence, the total cost is $O(nl + ml)$.
- Cost of **ClientDec&Ver**. Computing
$$\mathbf{Z} = \mathbf{P}_1^{-1}\left(\cdots\left(\mathbf{U}_1^{-1}\left(\mathbf{Z}''\mathbf{W}_1\cdots\mathbf{W}_{m-1}\right)\right)\cdots\right)\mathbf{P}_3$$
needs to perform at most $(4n(m-1) + 4m(n-1))$M, and the verification process needs to perform at most (λnl)M for some given constant λ. Hence, the total cost is $O(mn+nl)$. So, the overall time cost on the client side is $O(mn + nl + ml)$.

(2) Client-side cost of algorithm $\mathcal{SOA}_{MIC}(\mathbf{X})$.

- Cost of **ClientEnc**. For any valid input matrices $\mathbf{X} \in \mathbb{F}_q^{n\times n}$, computing
$$\mathbf{X}'' = \mathbf{U}_1\left(\cdots(\mathbf{U}_{n-1}(\mathbf{P}_1\mathbf{X}\mathbf{P}_2)\mathbf{V}_{n-1})\cdots\right)\mathbf{V}_1$$
needs to perform at most $8n(n-1)$M.
- Cost of **ClientDec&Ver**. Computing
$$\mathbf{Z} = \mathbf{P}_2\left(\mathbf{V}_{n-1}\left(\cdots\left(\mathbf{V}_1\mathbf{Z}''\mathbf{U}_1\right)\cdots\right)\mathbf{U}_{n-1}\right)\mathbf{P}_1$$
needs to perform at most $8n(n-1)$M, and the verification process needs to perform at most λn^2M for some given constant λ. Hence, the total cost is $O(n^2)$. So, the overall time cost on the client side is $O(n^2)$.

(3) Client-side cost of algorithm $\mathcal{SOA}_{MDC}(\mathbf{X})$.

- Cost of **ClientEnc**. For any valid input matrices $\mathbf{X} \in \mathbb{F}_q^{n\times n}$, computing
$$\mathbf{X}'' = \mathbf{U}_1\left(\cdots(\mathbf{U}_{n-1}(\mathbf{P}_1\mathbf{X}\mathbf{P}_2)\mathbf{V}_{n-1})\cdots\right)\mathbf{V}_1$$
needs to perform at most $(8n(n-1) + 2n^2)$ M$= O(n^2)$M.
- Cost of **ClientDec&Ver**. computing $\mathbf{e}_i = \mathbf{P} \times (\mathbf{L} \times (\mathbf{U} \times \mathbf{r}_i)) - \mathbf{X} \times \mathbf{r}_i$ needs to perform at most $\frac{n(n+1)}{2}$M, and thereby the verification process needs to perform at most $\lambda\frac{n(n+1)}{2}$M for some given constant λ. In addition, computing
$$\det(\mathbf{X}) = \frac{\det(\mathbf{P})(\prod_{i=1}^{n}\ell_{ii}u_{ii})}{\mathrm{sgn}(\pi_1)\prod_{i=1}^{n}\alpha_i \times \mathrm{sgn}(\pi_2)\prod_{i=1}^{n}\beta_i}$$
needs to perform at most $2(2n-1)$M. Hence, the total cost is $O(n^2)$. So, the overall time cost on the client side is $O(n^2)$.

In summary, we list our analysis results in Table 1, where t_{original}, t_{client1} and t_{client2} denote the client's time cost of accomplishing the original matrix operation on its own, the client-side time cost of encrypting the input matrices and the client-side time cost of decrypting and verifying the returned results from the cloud, respectively, and $t_{\text{client}} = t_{\text{client1}} + t_{\text{client2}}$. □

B. EXPERIMENTAL ANALYSIS

Besides the theoretical analysis, we also provide the experimental evaluation on the practical performance of our proposed algorithms. Our experiments are carried out on Ubuntu machine with 2.70 GHz Intel Pentium CPU G630 and 4 GB memory. We implement the proposed algorithms in Matlab 2017a. Take the field $F_q = F_{251}$ and denote t_{original}, t_{client1}, t_{client2} and t_{client} as the same meanings shown in the proof of Theorem 4.

Table 2, Table 3 and Table 4 present the experimental results on MMC, MIC and MDC respectively. Among them, all the input matrices are square and the dimension n varies from 100 to 3000, we choose $l = 50$ in the sub-algorithm **ClientDec&Ver** to balance the cheating resistance and the efficiency. To visualize our experimental results, we further convert the data in tables into 2-dimensional curves. That is, FIGURE2, FIGURE 3 and FIGURE 4 compare the client-side time costs of the proposed outsourcing algorithms (t_{client}) with those of the corresponding algorithms without outsourcing (t_{original}) respectively. To highlight the slope of the curve corresponding to t_{client}, the y-axis (vertical direction) in these figures is labeled with non-uniform scale. Finally, FIGURE 5 shows the client-side speedup ($t_{\text{original}}/t_{client}$). It can be observed from the tables and figures that our outsourcing algorithms achieve remarkable computational savings on the client side and are more effective as the dimension n of the input matrices increasing.

TABLE 1. Time cost of the proposed algorithms.

\	t_{client1}	t_{client2}	t_{client}	t_{original}	$t_{\text{original}}/t_{\text{client}}$
$\mathcal{SOA}_{MMC}(\mathbf{X},\mathbf{Y})$	$O(nl+ml)$	$O(mn+nl)$	$O(mn+nl+ml)$	$O(mln)$	$O(\frac{mln}{mn+nl+ml})$
$\mathcal{SOA}_{MMC}(\mathbf{X},\mathbf{Y})$ (when $m=l=n$)	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^{2.3728639})$	$O(n^{0.3728639})$
$\mathcal{SOA}_{MIC}(\mathbf{X})$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^{2.3728639})$	$O(n^{0.3728639})$
$\mathcal{SOA}_{MDC}(\mathbf{X})$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^{2.3728639})$	$O(n^{0.3728639})$

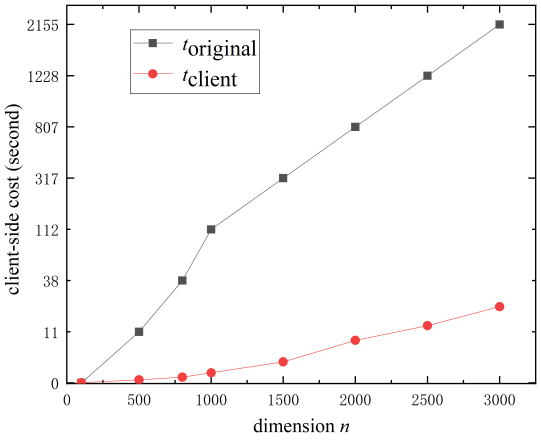


FIGURE 2. Client-side cost comparison of experimental results between $\mathcal{SOA}_{MMC}(\cdot)$ and algorithm without outsourcing.

TABLE 2. The experimental results of $\mathcal{SOA}_{MMC}(\cdot)$.

dimension n	t_{original}	t_{client1}	t_{client2}	t_{client}	$t_{\text{original}}/t_{\text{client}}$
100	0.1264	0.0688	0.1258	0.1946	0.6496
500	11.7416	0.3832	0.4109	0.7941	14.7860
800	38.2829	0.6572	0.7100	1.3672	28.0009
1000	112.5733	1.1096	1.1905	2.3000	48.9444
1500	316.9428	2.5234	2.0949	4.6182	68.6290
2000	806.9050	5.3848	3.7857	9.1705	87.9896
2500	1227.6657	8.0589	6.1739	14.2329	86.2553
3000	2155.1381	13.5576	10.6912	24.2487	88.8763

TABLE 3. The experimental results of $\mathcal{SOA}_{MIC}(\cdot)$.

dimension n	t_{original}	t_{client1}	t_{client2}	t_{client}	$t_{\text{original}}/t_{\text{client}}$
100	0.0538	0.0330	0.0844	0.1174	0.4580
500	6.7062	0.1837	0.2858	0.4694	14.2852
800	40.9169	0.3867	0.5869	0.9737	42.0229
1000	65.5457	0.5050	0.6810	1.1861	55.2634
1500	253.1054	1.1729	1.7369	2.9098	86.9847
2000	763.5120	2.4727	3.0928	5.5657	137.1818
2500	1319.2255	3.9808	5.0286	9.0095	146.4261
3000	2497.4030	6.5814	8.4534	15.0348	166.1078

VII. COMPREHENSIVE COMPARISONS WITH PRIOR WORK

In this section, we compare our algorithms with the existing schemes introduced in Section I-A. Let $\mathbb{R}$, $\mathbb{Z}$ denote the set of reals and the set of integers respectively, and let $\mathbb{Z}_p$ denote the prime field that is the finite field with a prime order p. Table 5 presents the comparison results of different schemes, out of which the first column denotes the outsourced matrix operations, the second column lists the existing schemes, the third column illustrates the type of input matrices, the fourth column (t_{Enc}) shows the client-side time overhead in the encryption stage, the fifth column ($t_{\text{Dec\&Ver}}$)

TABLE 4. The experimental results of $\mathcal{SOA}_{MDC}(\cdot)$.

dimension n	t_{original}	t_{client1}	t_{client2}	t_{client}	$t_{\text{original}}/t_{\text{client}}$
500	2.6328	0.1526	0.1267	0.2793	9.4253
800	10.9389	0.3069	0.3188	0.6257	17.4809
1000	23.7653	0.4389	0.4034	0.8423	28.2115
1500	68.7410	1.0764	0.9844	2.0608	33.3564
2000	169.8196	2.0187	1.8976	3.9163	43.3618
2500	338.7079	3.5441	2.8587	6.4028	52.8994
3000	614.4862	6.0895	4.6644	10.7539	57.1403
3500	959.1242	9.3445	6.0673	15.4118	62.2327

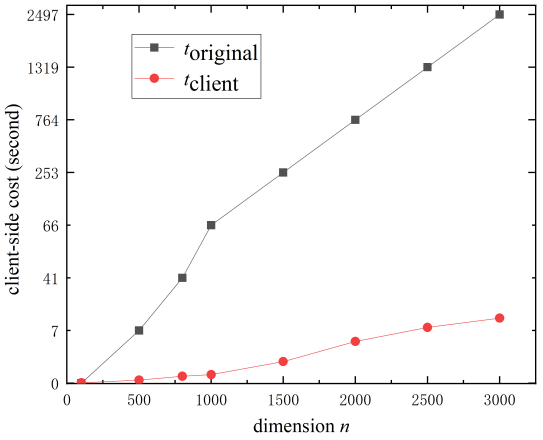


FIGURE 3. Client-side cost comparison of experimental results between $\mathcal{SOA}_{MIC}(\cdot)$ and algorithm without outsourcing.

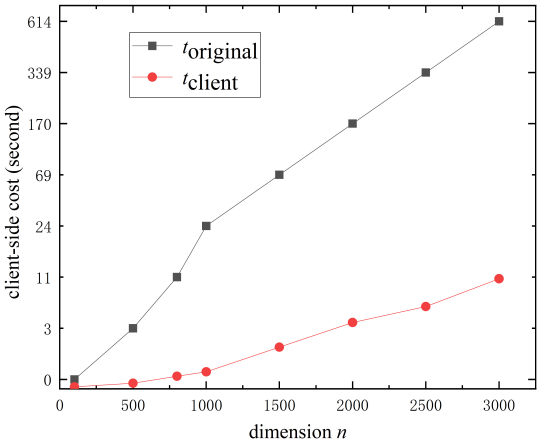


FIGURE 4. Client-side cost comparison of experimental results between $\mathcal{SOA}_{MDC}(\cdot)$ and algorithm without outsourcing.

TABLE 5. Comprehensive comparisons of the existing schemes mentioned in Sec.I-A.

Operation	Scheme	Matrix type	t_{Enc}	$t_{\text{Dec\&Ver}}$	t_{Cloud}	The number of zeros	The number of servers	HE
MMC	Benjamin et al [5]	$\mathbf{X}, \mathbf{Y} \in \mathbb{Z}^{n\times n}$	$O(n^2)$	$O(n^2)$	$O(n^3)$	protected	two	yes
	Atallah et al [2]	$\mathbf{X}, \mathbf{Y} \in \mathbb{Z}_p^{n\times n}$	$O(t^2n^2)$	$O(n^2)$	$O(tn^3)$	not protected	one or two	no
	Lei et al [27]	$\mathbf{X} \in \mathbb{R}^{m\times n}, \mathbf{Y} \in \mathbb{R}^{n\times s}$	$O(mn+ns)$	$O(ms)$	$O(msn)$	not protected	one	no
	Mohassel [32]	$\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{n\times n}$ (or $\mathbb{Z}_p^{n\times n}$)	$O(n^2)$	$O(n^2)$	$O(n^3)$	protected	one	yes
	Fu et al [21]	$\mathbf{X} \in \mathbb{R}^{m\times n}, \mathbf{Y} \in \mathbb{R}^{n\times s}$	$O(k(mn + ns))$	$O(mn + ms + ns)$	$O(mns)$	protected	one	no
	Our work	$\mathbf{X} \in \mathbb{F}_q^{m\times l}, \mathbf{Y} \in \mathbb{F}_q^{l\times n}$	$O(ml + ln)$	$O(mn+nl)$	$O(mnl)$	protected		
MIC	Lei et al [29]	$\mathbf{X} \in \mathbb{R}^{n\times n}$	$O(n^2)$	$O(n^2)$	$O(n^3)$	not protected	one	no
	Mohassel [32]	$\mathbf{X} \in \mathbb{R}^{n\times n}$	$O(n^2 \log n)$	$O(n^2 \log n)$	$O(n^3)$	protected	one	yes
	Our work	$\mathbf{X} \in \mathbb{F}_q^{n\times n}$	$O(n^2)$	$O(n^2)$	$O(n^3)$	protected	one	no
MDC	Lei et al [28]	$\mathbf{X} \in \mathbb{R}^{n\times n}$	$O((m + n)^2)$	$O((m + n)^2)$	$O((m + n)^3)$	protected	one	no
	Our work	$\mathbf{X} \in \mathbb{F}_q^{n\times n}$	$O(n^2)$	$O(n^2)$	$O(n^3)$	protected	one	no

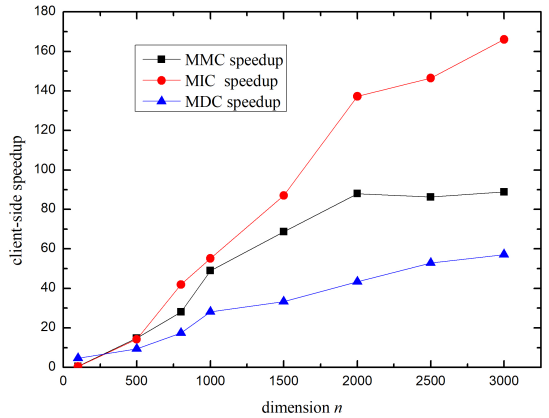


FIGURE 5. The speedup of client-side cost obtained by our proposed algorithms $\mathcal{SOA}_{\mathcal{MMC}}(\cdot)$, $\mathcal{SOA}_{\mathcal{MIC}}(\cdot)$ and $\mathcal{SOA}_{\mathcal{MDC}}(\cdot)$.

shows the client-side time overhead in the decryption and verification stage, the sixth column (t_{Cloud}) shows the cloud-side time overhead with basic algorithms for MMC, MIC and MDC in ciphertext matrices, the seventh column elucidates whether the number of zeros is protected in different schemes, the eighth column indicates the number of cloud servers in different schemes, and the last column clarifies whether the expensive homomorphic encryption operations are needed in different schemes.

VIII. CONCLUSION AND DISCUSSION

In this paper, we design three algorithms for outsourcing MMC, MIC and MDC to a malicious cloud. Theoretical analysis and experimental evaluation indicate that the proposed algorithms achieve a better tradeoff between the privacy and the efficiency. The main technique involving in our algorithm is a novel matrix encryption/decryption method which encrypts/decrypts a matrix by consecutive and sparse unimodular matrix transformations. A natural question is whether the similar technique can be used for outsorucing matrix operations over the field of reals $\mathbb{R}$, the field of rationals $\mathbb{Q}$ or the ring of integers $\mathbb{Z}$. Our experimental tests indicate that the answer is probably not. Since $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{Z}$ are unbounded, the blow-up of the size of entries in the product matrix may be very fast, which results in the elements in the final ciphertext matrix are extremely large and thereby the similar encryption algorithm become inefficient. Another interesting problem is applying similar technique to outsource other common matrix operations, such as matrix rank computation, matrix factorization, matrix's characteristic polynomial and eigenvalues computation etc, which is left for future work.

REFERENCES

[1] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange—A new hope." in *Proc. 25th USENIX Secur. Symp.*, Austin, TX, USA, 2016, pp. 327–343.

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

[3] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. E. Spafford, "Secure outsourcing of scientific computations," *Adv. Comput.*, vol. 54, pp. 215–272, Jan. 2002.

[4] Z. Bai, G. Fahey, and G. Golub, "Some large-scale matrix computation problems," *J. Comput. Appl. Math.*, vol. 74, nos. 1–2, pp. 71–89, Nov. 1996.

[5] 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.

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

[7] R. Chandra, *Permutation and Combinations*. Chapman and Hall: London, U.K., 2016.

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

[9] K. Bryan and T. Leise, "The $25,000,000,000 eigenvector: The linear algebra behind Google," *SIAM Rev.*, vol. 48, no. 3, pp. 569–581, Aug. 2006.

[10] B. Carpentieri, "Sparse preconditioners for dense linear systems from electromagnetic applications," Ph.D. dissertation, CERFACS, Toulouse, France, 2002.

[11] 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.

[12] X. Chen, J. Li, X. Huang, J. Li, Y. Xiang, and D. S. Wong, "Secure outsourced attribute-based signatures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3285–3294, Dec. 2014.

[13] X. Chen, J. Li, X. Huang, J. Ma, and W. Lou, "New publicly verifiable databases with efficient updates," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 5, pp. 546–556, Sep./Oct. 2015.

[14] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2386–2396, Sep. 2014.

[15] X. Chen, J. Li, J. Weng, J. Ma, and W. Lou, "Verifiable computation over large database with incremental updates," *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 3184–3195, Oct. 2016.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

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

[18] K. Elkhiyaoui, M. Önen, M. Azraoui, and R. Molva, "Efficient techniques for publicly verifiable delegation of computation," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, New York, NY, USA, Jun. 2016, pp. 119–128.

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

[20] R. Fisher and F. Yates, *Statistical Tables for Biological, Agricultural and Medical Research*. Oliver and Boyd: Edinburgh, U.K., 1953.

[21] S. Fu, Y. Yu, and M. Xu, "A secure algorithm for outsourcing matrix multiplication computation in the cloud," in *Proc. 5th ACM Int. Workshop Secur. Cloud Comput.*, Apr. 2017, pp. 27–33.

[22] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2011, pp. 69–77.

[23] 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.

[24] X. Hu and C. Tang, "Secure outsourced computation of the characteristic polynomial and eigenvalues of matrix," *J. Cloud Comput.*, vol. 4, no. 1, p. 7, Dec. 2015.

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

[26] F. L. Gall, "Powers of tensors and fast matrix multiplication," in *Proc. 39th Int. Symp. Symbolic Algebr. Comput.*, New York, NY, USA, Jul. 2014, pp. 296–303.

[27] 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.

[28] X. Lei, X. Liao, T. Huang, and H. Li, "Cloud computing service: The case of large matrix determinant computation," *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 688–700, Sep./Oct. 2015.

[29] 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.

[30] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *Deep Space Netw. Prog. Rep.*, vol. 44, pp. 114–116, Jan./Feb. 1978.

[31] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*. Boston, MA, USA: Kluwer Academic Publishers, 1987.

[32] P. Mohassel, "Efficient and secure delegation of linear algebra," in *Proc. IACR Cryptol. ePrint Arch.*, Jan. 2011, p. 605.

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

[34] P. Okunev and C. R. Johnson. (Jan. 2005). "Necessary and sufficient conditions for existence of the LU factorization of an arbitrary matrix." [Online]. Available: https://arxiv.org/abs/math/0506382

[35] 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.

[36] V. Pan and J. Reif, "Efficient parallel solution of linear systems," in *Proc. 17th Annu. ACM Symp. Theory Comput.*, May 1985, pp. 143–152.

[37] B. Paul and R. L. Huston, "Kinematics and dynamics of planar machinery," *J. Appl. Mech.*, vol. 47, no. 2, p. 459, Jun. 1980.

[38] R. Roth, *Introduction to Coding Theory*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[39] S. Salinas, X. Chen, J. Ji, and P. A. Li, "A tutorial on secure outsourcing of large-scale computations for big data," *IEEE Access*, vol. 4, pp. 1406–1416, 2016.

[40] 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./May 2015, pp. 1035–1043.

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

[42] Z. Shan, K. Ren, M. Blanton, and C. Wang, "Practical secure computation outsourcing: A survey," *Acm Comput. Surv.*, vol. 51, no. 2, Jun. 2018, Art. no. 31.

[43] G. Sheng, C. Tang, W. Gao, and Y. Yin, "MD-VC$_{Matrix}$: An efficient scheme for publicly verifiable computation of outsourced matrix multiplication," in *Network and System Security*, J. Chen, V. Piuri, C. Su, and M. Yung, Ed. Cham, Switzerland: Springer, 2016, pp. 349–362.

[44] Q. Su, J. Yu, C. Tian, H. Zhang, and R. Hao "How to securely outsource the inversion modulo a large composite number, *J. Syst. Softw.*, vol. 129, p. 26–34, Jul. 2017.

[45] J. P. Tiemstra, *The Foundations of Positive and Normative Economics: A Handbook*, vol. 68, London, U.K.: Oxford Univ. Press, 2008, no. 3, pp. 377–380.

[46] S. Zhang, H. Li, Y. Dai, J. Li, M. He, and R. Lu, "Verifiable outsourcing computation for matrix multiplication with improved efficiency and applicability," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 5076–5088, Dec. 2018.

[47] S. Zhang, H. Li, K. Jia, Y. Dai, and L. Zhao, "Efficient secure outsourcing computation of matrix multiplication in cloud computing," in *Proc. IEEE Global Commun. Conf.*, Dec. 2016, pp. 1–6.

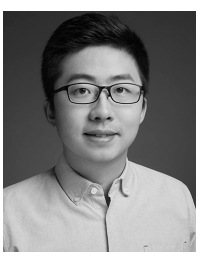
[48] X. Zhang, T. Jiang, K.-C. Li, A. Castiglione, and X. Chen, "New publicly verifiable computation for batch matrix multiplication, *Inf. Sci.*, vol. 479, pp. 664–678, Apr. 2019.

[49] Y. Zhang and M. Blanton, "Efficient secure and verifiable outsourcing of matrix multiplications," in *Proc. Int. Conf. Inf. Secur.*, Oct. 2014, pp. 158–178.

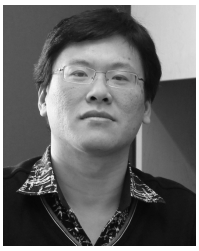
[50] L. Zhou and C. Li "Outsourcing eigen-decomposition and singular value decomposition of large matrix to a public cloud," *IEEE Access*, vol. 4, pp. 869–879, 2017.

SHENGXIA ZHANG received the B.E. degree in computer science and technology from Hexi University, in 2016. 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 outsourcing computation, 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 at the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing. He is currently an Assistant Professor with the College of Computer Science and Technology, Qingdao University, Qingdao, China. His research interests include lattice-based cryptography and cloud computing security.



HANLIN ZHANG received the B.S. degree in software engineering from Qingdao University, in 2010, and the M.S. degree in applied information technology and the Ph.D. degree in information technology from Towson University, MD, USA, in 2011 and 2016, respectively. He is currently an Assistant Professor with the College of Computer Science and Technology, Qingdao University. His research interests include cloud computing security, blockchain technology, and the IoT security.



JIA YU received the B.S. and M.S. degrees from the School of Computer Science and Technology, Shandong University, in 2003 and 2000, respectively, and the Ph.D. degree from the Institute of Network Security, Shandong University, in 2006. He was a Visiting Professor with the Department of Computer Science and Engineering, The State University of New York at Buffalo, from 2013 to 2014. He is currently a Professor with the College of Computer Science and Technology, Qingdao University, Qingdao, China. His research interests include cloud computing security, key evolving cryptography, digital signature, and network security.

FENGJUN LI received the B.E. degree (Hons.) from the University of Science and Technology of China, in 2001, the M.Phil. degree from The Chinese University of Hong Kong, in 2004, and the Ph.D. degree from the Pennsylvania State University, in 2010. She is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, The University of Kansas. Her research interests include the broad areas of security and privacy for distributed information systems, cyber-physical systems, and communication networks. She received the Kansas NSF EPSCoR First Award, in 2014, and the Miller Scholar Award from The University of Kansas, in 2016.