Cloud Computing Service: The Case of Large Matrix Determinant Computation

Xinyu Lei, Xiaofeng Liao, *Senior Member, IEEE*, Tingwen Huang, and Huaqing Li

Abstract—Cloud computing paradigm provides an alternative and economical service for resource-constrained clients to perform large-scale data computation. Since large matrix determinant computation (DC) is ubiquitous in the fields of science and engineering, a first step is taken in this paper to design a protocol that enables clients to securely, verifiably, and efficiently outsource DC to a malicious cloud. The main idea to protect the privacy is employing some transformations on the original matrix to get an encrypted matrix which is sent to the cloud; and then transforming the result returned from the cloud to get the correct determinant of the original matrix. Afterwards, a randomized Monte Carlo verification algorithm with one-sided error is introduced, whose superiority in designing inexpensive result verification algorithm for secure outsourcing is well demonstrated. In addition, it is analytically shown that the proposed protocol simultaneously fulfills the goals of correctness, security, robust cheating resistance, and high-efficiency. Extensive theoretical analysis and experimental evaluation also show its high-efficiency and immediate practicability. It is hoped that the proposed protocol can shed light in designing other novel secure outsourcing protocols, and inspire powerful companies and working groups to finish the programming of the demanded all-inclusive scientific computations outsourcing software system. It is believed that such software system can be profitable by means of providing large-scale scientific computation services for so many potential clients.

Index Terms—Cloud computing, secure outsourcing, determinant computation, Monte Carlo verification, outsourcing software system

1 Introduction

Large-scale data computation entails a huge commitment of computing resources, making it prohibitive to resource-constrained clients. Fortunately, cloud computing paradigm provides an approach for small to medium size business to perform large-scale data processing [1]. The relationship between large-scale data computation and cloud computing service seems to be inherent: the former becomes the service object of the latter, while the latter offers an alternative solution for the former. It is witnessed that cloud computing has become increasingly important to provide service-oriented large-scale data computation in third-party data management settings. For instance, cloud computing services [2], [3], [4], [5], [6], [7] have been successful implemented in several information systems in recent years.

According to the Cloud Services User Survey and CSA Security Guidance [8], [9], security has been identified as the top issue that prevents the business intelligence from taking advantage of cloud services. Yet, the cloud security technique is under developed in comparison with other cloud-related techniques and hence it is becoming the bottleneck. To some extent, security concern is now driving how we define and develop cloud computing solutions.

• *X. Lei, X. Liao, and H. Li are with the State Key Laboratory of Power Transmission Equipment and System Security and New Technology, College of Computer Science, Chongqing University, Chongqing 400044, P. R., China.*
E-mail: xy-lei@qq.com, xfliao@cqu.edu.cn, lhq_jsack@126.com.
• *T. Huang is with Texas A&M University at Qatar, Doha PO Box 23874, Qatar. E-mail: tingwen.huang@qatar.tamu.edu.*

Manuscript received 12 Feb. 2014; revised 28 May 2014; accepted 6 June 2014. Date of publication 17 June 2014; date of current version 9 Oct. 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2014.2331694

Outsourcing of scientific computation is identified as an important application of cloud computing. By virtue of harnessing cloud computing, the resource-constrained clients can off-load their computation-intensive tasks to powerful clouds. In contrast to setting up and maintaining their own infrastructures, the clients can economically share the massive computational power, storage, and even some softwares of the clouds. Hence, this promising application well captures some well-known service notions of cloud computing like Network-as-a-Service (NaaS), Infrastructure-as-a-Service (IaaS), and Software-as-a-Service (SaaS).

1.1 Challenges

Although it is quite promising, when the outsourcing scientific computation meets the security concerns, three major challenges arise. More specifically, the first challenge is the client's input/output data privacy. The outsourced computational problems and their results often contain sensitive information, such as the business financial records, VIP customers lists, engineering data, or proprietary asset data, etc. To hide these information from the cloud, clients need to encrypt their data before outsourcing and decrypt the returned result from the cloud after outsourcing. The second challenge is the verification of the result returned by the cloud. A cloud server might not always provide the accurate result of a given computational task. As an example of intentional reasons, for the outsourced computational intensive tasks, there are strong financial incentives for the cloud to be lazy and just return incorrect answers to the client if such answers require less work and are unlikely to be detected by the client. Besides, some accidental reasons such as possible software bugs or hardware failures may also result in wrong computational results. Consequently, the outsourcing protocol must be designed in such a way

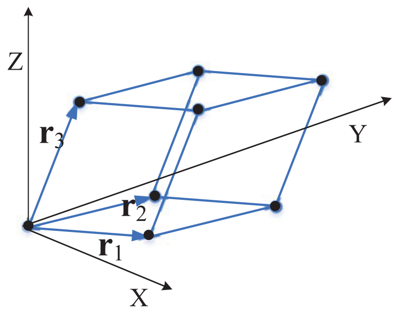


Fig. 1. A Parallelepiped Formed by Vectors $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$.

that it is able to detect whether the returned result is correct. The third challenge is efficiency. On one hand, a key requirement is that the amount of local work performed by the client must be substantially cheaper than performing the original computational problem on its own. Otherwise, it does not make sense for the client to resort to the cloud. On the other hand, it is also desirable to maintain the amount of work performed by the cloud as close as possible to that needed to compute the original problem by the client itself. Otherwise, the cloud may be unable to complete the task in a reasonable amount of time, or the cost of the cloud may become prohibitive. To summarize, a protocol for computation outsourcing should satisfy the following for aspects: correctness, security, verifiability and efficiency.

1.2 Motivations

Determinant computation (DC) is a basic computational problem in scientific and engineering fields and has a number of applications. First of all, DC can be used to compute the hyper volume [10].

Example 1. Take the simple visual 3D parallelepiped as an example, suppose that the parallelepiped in Fig. 1 is formed by vectors $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$, where $\mathbf{r}_1 = (3, 1, 1)$, $\mathbf{r}_2 = (1, 3, 1)$, and $\mathbf{r}_3 = (1, 1, 3)$, then the volume of the parallelepiped can be computed by

$$V = \det \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix} = 20.$$

This formula can be used in computing hyper volume in high-dimensional space where the dimensions are far more than 3.

Second, DC provides a way to solve a system of linear equations by virtue of the so-called Cramer's rule [11].

Example 2. Consider a system of n linear equations for n unknowns, represented in matrix multiplication form as follows: $\mathbf{Ax} = \mathbf{b}$, where the $n \times n$ matrix has a nonzero determinant, and the vector $\mathbf{b}$ is the column vector of the variables. Then the Cramer's rule states that the values for the unknowns are given by:

$$x_i = \frac{\det(\mathbf{A}_i)}{\det(\mathbf{A})}, i = 1, \ldots, n,$$

where $\mathbf{A}_i$ is the matrix formed by replacing the ith column of $\mathbf{A}$ by the column vector $\mathbf{b}$.

Furthermore, DC is well rooted in other scientific applications including mathematical physics analysis [12], Lagrange interpolation analysis [13], anharmonic oscillator analysis [14], to just list a few. When the restricted computational resources are possessed by these clients and DC deals with a large matrix (or a batch of large matrices), an economical solution is to outsource DC to a powerful cloud. Even if the data is in a moderate scale, for clients as battery-limited mobile phones, portable devices, or embedded smart cards, secure outsourcing of DC is preferred. Consequently, we are motivated to design a protocol that enables clients to securely, verifiably, and efficiently outsource DC to a cloud.

1.3 Contributions

In this paper, we take a first step forward to address the issue of how to outsource DC to a remote malicious cloud server while ensuring protocol correctness, maintaining data input/output privacy, realizing result verifiability, and improving computational efficiency. From the complexity point of view, the hitherto best algorithm known for DC shares the same time complexity of matrix multiplication, i.e., $O(n^{2.373})$. An important challenge is hence to ensure the local computation performed by the client is less than $O(n^{2.373})$. By means of introducing the block matrix and permutation technique to protect privacy and adopting LU decomposition and Monte Carlo technique to handle result verification, the local work of the client can be restricted to $O(n^2)$. That is to say, the client can reduce its original $O(n^{2.373})$ work to $O(n^2)$ work by outsourcing DC to a cloud. Moreover, experimental evaluation is also provided to demonstrate that the proposed protocol is able to allow the client to outsource DC to a cloud and gain substantial computation savings.

1.4 Organization

The remainder of this paper proceeds as follows. Section 2 introduces some essential preliminaries. In Section 3, we describe our protocol with detailed techniques. Sections 4 and 5 give some related analysis and performance evaluation, followed by Section 6 which overviews the related work. Finally, some conclusions are drawn in Section 7.

2 Preliminaries

2.1 System Model, Threat Model, Design Goals, and Framework

2.1.1 System Model

We consider the secure DC outsourcing system model, as illustrated in Fig. 2. A client with low computational power intends to outsource the original DC to a cloud service provider, who has massive computational power and special softwares. In order to protect input privacy, the client encrypts the original DC using a secret key K to get a new DC problem, written as DC_K. Later, the encrypted DC_K is given to the cloud for a result. Once the cloud receives DC_K, the computation is carried out with softwares; then the cloud sends back the result to DC_K. The cloud also sends back a proof Γ that tries to prove the returned result is indeed correct and the cloud does not cheat. On receiving the returned result, the client decrypts it using the secret key K to get the result to the original DC. Meanwhile, the

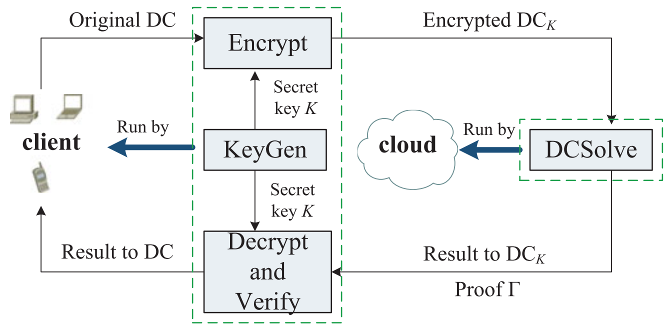


Fig. 2. Secure DC Outsourcing System Model.

client checks whether the result is correct: if yes, accepts it; otherwise, just rejects it.

2.1.2 *Threat Model*

The security threats faced by the outsourcing system model primarily come from the behavior of the cloud. Generally, there are two levels of threat models in outsourcing: semi-honest cloud model and malicious cloud model [15]. In the semi-honest cloud model, the cloud correctly follow the protocol specification. However, the cloud records all the information it can access, and attempts to use this to learn information that should remain private. While in the malicious cloud model, the cloud can arbitrarily deviate from the protocol specification. The malicious cloud may just return a random result to the client to save its computing resources, while hoping not to be detected by the client. Therefore, an outsourcing protocol in the malicious cloud model should be able to handle result verification. In this paper, we assume that the cloud is malicious. Our protocol should be able to resist such a malicious cloud.

2.1.3 *Design Goals*

We identify four goals that the outsourcing protocol should satisfy. 1) *Correctness.* If both the client and the cloud follow the protocol honestly, the DC can be indeed fulfilled by the cloud and the client gets the correct result to the original DC. 2) *Security.* The protocol can protect the privacy of the client's data. On one hand, given the encrypted DC_K problem, the cloud cannot get meaningful knowledge of the client's input data, which is referred to as *input privacy*. On the other hand, the correct result to the original DC is also hidden from the cloud, and this is called as *output privacy*. The input/output privacy should be protected "as-strong-as-possible". 3) *Robust cheating resistance.* The correct result from a faithful cloud server must be verified successfully by the client. No false result from a cheating cloud server can pass the verification with a non-negligible probability. 4) *Efficiency.* The local computation done by the client should be substantially less than the computation of the original DC on its own. In addition, the amount of computation on computing the encrypted DC_K should be as close as possible to that on computing the original DC.

2.1.4 *Framework*

Syntactically, a secure DC outsourcing protocol should contain five sub-algorithms: 1) the algorithm for key generation KeyGen, 2) the algorithm for DC encryption DCEnc, 3) the algorithm for solving DC_K problem DCSolve, 4) the algorithm for DC decryption DCDec, and 5) the algorithm for result verification ResultVerify. One significant difference between this framework and the traditional encryption framework is that in this case both encryption and decryption process occur in the client side. This eliminates the expensive public key exchange process in the traditional encryption framework. Therefore, this framework is able to efficiently realize one-time-pad type of flexibility. That is to say, KeyGen will be run every time for a new outsourced matrix instance to enhance security. Once we have this framework, we just need to work out the details of these five sub-algorithms, which will be shown in Section 3.

2.2 Mathematical Background

Permutation function is well studied in group theory and combinatorics [16]. Using Cauchy's two-line notation, a permutation function can be written as

$$\begin{pmatrix} 1 & \cdots & n \\ p_1 & \cdots & p_n \end{pmatrix}, \tag{1}$$

where one lists the preimage element in the first row, and for each preimage element lists its image under the permutation below it in the second row. A permutation can also be represented by *cycle notation* and *transposition notation*[16]. An illustrative example is given as follows.

Example 3.

$$\begin{aligned}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 2 & 1 \end{pmatrix} &= (1\ 3\ 5)(2\ 4)\ (\textbf{Cycle Notation}) \\ &= (1\ 5)(1\ 3)(2\ 4)(\textbf{Transposition Notation}).\end{aligned} \tag{2}$$

This paper defaultly applies the one-line array $\pi(i) = p_i$, where $i = 1, \ldots, n$, to denote (1). Let π^{-1} denote the inverse function of π. As shown in Example 3, a permutation can be represented by a sequence of *transpositions* (two-element exchanges). It is defined as *even permutation* if it consists of an even number of transpositions and *odd permutation* if it consists of an odd number of transpositions. The *sign* of a permutation is denoted as $\mathrm{sgn}(\pi)$ and defined as

$$\mathrm{sgn}(\pi) = \begin{cases} +1, & \text{if } \pi \text{ is even}, \\ -1, & \text{if } \pi \text{ is odd}. \end{cases} \tag{3}$$

What's more, the Kronecker delta function [17] $\delta_{x,y}$ equals 1 if $x = y$ and 0 if $x \neq y$. Shown in Table 1 are the summarization of main terms throughout this paper.

Two important properties regarding to the determinant of matrix are given in the following lemmas [11].

Lemma 1. *A row transposition (two-row exchange) or column transposition (two-column exchange) of a matrix changes the sign of its determinant.*

Lemma 2. *Suppose that* **A**, **B**, **0**, *and* **D** *are matrices of dimension* $n \times n$, $n \times m$, $m \times n$, *and* $m \times m$, *respectively, then*

TABLE 1
Terms and Description

Terms	Description
π^{-1}	the inversion of the permutation π
$\mathrm{sgn}(\pi)$	the sign of the permutation π
$\delta_{x,y}$	the Kronecker delta function
$\mathbf{X}$	a full rank matrix of order $n \times n$
$\mathbf{X}(i,j)$, $\mathbf{X}_{ij}$, $x_{i,j}$, or x_{ij}	the entry in ith row and jth column in matrix $\mathbf{X}$
$\det(\mathbf{X})$	the determinant of matrix $\mathbf{X}$
D_Y, D_X	the unchecked determinant of $\mathbf{Y}$ and $\mathbf{X}$ in Algorithm 5 (Procedure DC-Decryption)
$\mathrm{diag}(d_1, \ldots, d_n)$	a diagonal matrix matrix with main diagonal entries being $d_1, \ldots, d_n$
$\alpha \leftarrow \mathcal{K}_a$	the process of choosing an element α from set $\mathcal{K}_\alpha$ uniformly and randomly
$Prob_1$	the probability of non-detection of false returned results in one round of the random check process in Algorithm 6 (Procedure Result-Verification)
$Prob_l$	the probability of non-detection of false returned results in whole l times random check processes in Algorithm 6 (Procedure Result-Verification)
$Prob_f$	the probability of *check failure*, i.e., the probability of non-detection of false returned results in Algorithm 6 (Procedure Result-Verification)

$$\det\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{D} \end{bmatrix} = \det(\mathbf{A})\det(\mathbf{D}). \tag{4}$$

3 Protocol Construction

In this section, each part of the framework for secure outsourcing of DC will be individually solved.

3.1 Secret Key Generation

Consider a full rank matrix $\mathbf{X} \in \mathbb{R}^{\mathbf{n}\times\mathbf{n}}$, the resource-constrained client intends to securely outsource the computation of $\det(\mathbf{X})$ to a powerful cloud. The protocol starts by invoking Algorithm 1 (Procedure Secret-Key-Generation) to set up a secret key K.

Algorithm 1. Procedure Secret-Key-Generation

```
Input:  A security parameter $\kappa$.
Output: Secret key $K$: $m$, $\{d_1, \ldots, d_m\}$, $\{\alpha_1, \ldots, \alpha_{n+m}\}$,
        $\{\beta_1, \ldots, \beta_{n+m}\}$, $\mathbf{B}$, $\pi_1$, $\pi_2$, $\mathrm{sgn}(\pi_1)$, $\mathrm{sgn}(\pi_2)$.
1: On input a security parameter $\kappa$, which specifies a positive integer $m$ and three key spaces $\mathcal{K}_d$, $\mathcal{K}_\alpha$, and
   $\mathcal{K}_\beta$, the client first picks three sets of random numbers:
   $\{d_1, \ldots, d_m\} \leftarrow \mathcal{K}_d$, $\{\alpha_1, \ldots, \alpha_{n+m}\} \leftarrow \mathcal{K}_\alpha$, $\{\beta_1, \ldots, \beta_{n+m}\} \leftarrow \mathcal{K}_\beta$, where $0 \notin \mathcal{K}_d \cup \mathcal{K}_\alpha \cup \mathcal{K}_\beta$.
2: Then, the client picks a matrix $\mathbf{B}$ of dimension $n \times m$ with the number of zero entries uniformly and ran-
   domly distributing in $\{0, \ldots, nm\}$.
3: Finally, the client invokes Algorithm 2 to generate two random permutations $\pi_1$ and $\pi_2$ of the integers
   $1, \ldots, n+m$ and records $\mathrm{sgn}(\pi_1)$ and $\mathrm{sgn}(\pi_2)$ as well.
```

Algorithm 2 is modified from Fisher-Yates shuffle [18], [19] by adding a parameter sgn to record the sign of the generated permutation. There are several variants of Fisher-Yates shuffle to generate a random permutation. Nevertheless, the asymptotic time complexity of Fisher-Yates shuffle has already been optimal. This is the reason why this algorithm could be used for random permutation generation in this work.

Algorithm 2. Random Permutation Generation

```
Input: $n + m$.
Output: $\pi$ and $\mathrm{sgn}(\pi)$.
 1: Set $\pi = \mathrm{I}_{n+m}$ (identical permutation);
 2: Set $sgn = +1$ (even permutation);
 3: for $i = n + m$ down to 2 do
 4:   Set $j$ to be a random integer with $1 \leq j \leq i$;
 5:   Swap $\pi[j]$ and $\pi[i]$;
 6:     if ($j! = i$ && $sgn == +1$) then
 7:       $sgn = -1$ (odd permutation);
 8:     end if
 9:     if ($j! = i$ && $sgn == -1$) then
10:       $sgn = +1$;
11:     end if
12: end for
```

3.2 DC Encryption

Next, we describe Algorithm 3 (Procedure DC-Encryption). The permutation technique described in this procedure is first used in [20] to address matrix multiplication outsourcing problem. This permutation technique is adopted as a subroutine in Algorithm 3 (Procedure DC-Encryption). Some novel crucial results of this technique regarding to DC outsourcing are identified in Lemma 3.

Algorithm 3. Procedure DC-Encryption

```
Input: The original matrix $\mathbf{X}$ and the secret key $K$: $m$,
       $\{d_1, \ldots, d_m\}$, $\{\alpha_1, \ldots, \alpha_{n+m}\}$, $\{\beta_1, \ldots, \beta_{n+m}\}$, $\mathbf{B}$,
       $\pi_1$, $\pi_2$, $\mathrm{sgn}(\pi_1)$, $\mathrm{sgn}(\pi_2)$.
Output: Matrix $\mathbf{Y}$.
1: The client generates a matrix $\mathbf{T} = [\begin{smallmatrix}\mathbf{X} & \mathbf{0} \\ \mathbf{B} & \mathbf{D}\end{smallmatrix}]$, where $\mathbf{0}$ is an
   $m \times n$ zero matrix and $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_m)$.
2: The client generates two $(m+n) \times (m+n)$ matrices
   $\mathbf{P}_1$ and $\mathbf{P}_2$, where $\mathbf{P}_1(i,j) = \alpha_i \delta_{\pi_1(i),j}$ and
   $\mathbf{P}_2(i,j) = \beta_i \delta_{\pi_2(i),j}$.
3: The client computes $\mathbf{Y} = \mathbf{P}_1 \mathbf{T} \mathbf{P}_2^{-1}$. According to
   Theorem 1, the client can use (7) to efficiently (via time
   $O(n^2)$) compute $\mathbf{Y}$.
```

```
4: Later, the encrypted matrix $\mathbf{Y}$ will be outsourced to
   the cloud.
```

Lemma 3. *In Algorithm 3 (Procedure DC-Encryption), the determinants of matrix* $\mathbf{P}_1$ *and* $\mathbf{P}_2$ *are given by*

$$\begin{cases} \det(\mathbf{P}_1) = \mathrm{sgn}(\pi_1) \prod_{i=1}^{n+m} \alpha_i, \\ \det(\mathbf{P}_2) = \mathrm{sgn}(\pi_2) \prod_{i=1}^{n+m} \beta_i. \end{cases} \tag{5}$$

Proof. It suffices to prove the case of $\mathbf{P}_1$. If $\mathbf{P}_1$ is generated by $\mathbf{P}_1(i,j) = \alpha_i \delta_{\pi_1(i),j}$, then the diagonal matrix $\mathbf{\Lambda}_1 = \mathrm{diag}(\alpha_1, \ldots, \alpha_{n+m})$ can be transformed to be $\mathbf{P}_1$ through a number of column transpositions. The parity of the number of the column transpositions is the same with the sign of permutation π_1. Recall Lemma 1, one has $\det(\mathbf{P}_1) = \mathrm{sgn}(\pi_1)\det(\mathbf{\Lambda}_1) = \mathrm{sgn}(\pi_1) \prod_{i=1}^{n+m} \alpha_i$. □

Two valid examples can help the readers to gain an insightful understanding of Lemma 3.

Example 4.

- *Case 1*. Suppose that $\{\alpha_1, \alpha_2, \alpha_3\} = \{1, 2, 3\}$ and $\pi_1 = \binom{1\,2\,3}{2\,1\,3} = (12)$, then

$$\mathbf{P}_1 = \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 3 \end{bmatrix}.$$

Let

$$\mathbf{\Lambda}_1 = \mathrm{diag}(\alpha_1, \alpha_2, \alpha_3) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix},$$

then $\mathbf{\Lambda}_1$ can be transformed to be $\mathbf{P}_1$ through exchanging the first column vector and second column vector, because of $\pi_1 = (12)$. The permutation π_1 consists of only one transposition, therefore it is an odd permutation, i.e., $\mathrm{sgn}(\pi_1) = -1$. It holds immediately that $\det(\mathbf{P}_1) = \mathrm{sgn}(\pi_1)\det(\mathbf{\Lambda}_1) = (-1) \times \prod_{i=1}^{3} \alpha_i = -6$.

- *Case 2*. If the above π_1 is replaced by an even permutation $\pi_1' = \binom{1\,2\,3}{2\,3\,1} = (13)(12)$, then

$$\mathbf{P}_1' = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 2 \\ 3 & 0 & 0 \end{bmatrix}.$$

In this case,

$$\mathbf{\Lambda}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

can be transformed to be $\mathbf{P}_1'$ through first exchanging the first column vector and the third column vector and then exchanging the the first column and the second column vector, because of $\pi_1' = (13)(12)$. It holds that $\mathrm{sgn}(\pi_1') = +1$, which results in $\det(\mathbf{P}_1') = \mathrm{sgn}(\pi_1')\det(\mathbf{\Lambda}_1) = (+1) \times \prod_{i=1}^{3} \alpha_i = 6$.

Lemma 4. *In Procedure Algorithm 3 (DC-Encryption), the matrix* $\mathbf{P_2}$ *is invertible. More precisely,*

$$\mathbf{P}_2^{-1}(i,j) = (\beta_j)^{-1} \delta_{\pi_2^{-1}(i),j}. \tag{6}$$

Proof. Since $0 \notin \mathcal{K}_\beta$, note from Lemma 3 that the determinant of $\mathbf{P_2}$ satisfies $\det(\mathbf{P}_2) = \mathrm{sgn}(\pi_2) \prod_{i=1}^{n} \beta_i \neq 0$, which demonstrates that $\mathbf{P_2}$ is invertible. The remaining proof is straightforward and therefore it is omitted. □

Theorem 1. *In Algorithm 3 (Procedure DC-Encryption), if* $\mathbf{Y} = \mathbf{P}_1\mathbf{T}\mathbf{P}_2^{-1}$, *then it holds that*

$$\mathbf{Y}(i,j) = (\alpha_i/\beta_j)\mathbf{T}(\pi_1(i), \pi_2(j)). \tag{7}$$

Proof. Let

$$\mathbf{T} = \begin{bmatrix} t_{1,1} & \cdots & t_{1,q} \\ \vdots & \ddots & \vdots \\ t_{q,1} & \ldots & t_{q,q} \end{bmatrix},$$

where $q = n + m$. Observe that $\mathbf{P}_1(i,j) = \alpha_i \delta_{\pi_1(i),j}$, this leads to

$$\mathbf{P}_1\mathbf{T} = \begin{bmatrix} \alpha_1 t_{\pi_1(1),1} & \cdots & \alpha_1 t_{\pi_1(1),q} \\ \vdots & \ddots & \vdots \\ \alpha_i t_{\pi_1(i),1} & \cdots & \alpha_i t_{\pi_1(i),q} \\ \vdots & \ddots & \vdots \\ \alpha_q t_{\pi_1(q),1} & \ldots & \alpha_q t_{\pi_1(q),q} \end{bmatrix}.$$

From Lemma 4, $\mathbf{P}_2^{-1}(i,j) = (\beta_j)^{-1} \delta_{\pi_2^{-1}(i),j}$, then one can obtain

$$\mathbf{P}_1\mathbf{T}\mathbf{P}_2^{-1} = \begin{bmatrix} \frac{\alpha_1}{\beta_1} t_{\pi_1(1),\pi_2(1)} & \cdots & \frac{\alpha_1}{\beta_j} t_{\pi_1(1),\pi_2(j)} & \cdots & \frac{\alpha_1}{\beta_q} t_{\pi_1(1),\pi_2(q)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\alpha_i}{\beta_1} t_{\pi_1(i),\pi_2(1)} & \cdots & \frac{\alpha_i}{\beta_j} t_{\pi_1(i),\pi_2(j)} & \cdots & \frac{\alpha_i}{\beta_q} t_{\pi_1(i),\pi_2(q)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\alpha_q}{\beta_1} t_{\pi_1(q),\pi_2(1)} & \cdots & \frac{\alpha_q}{\beta_j} t_{\pi_1(q),\pi_2(j)} & \cdots & \frac{\alpha_q}{\beta_q} t_{\pi_1(q),\pi_2(q)} \end{bmatrix}.$$

This can be finally rewritten as $\mathbf{P}_1\mathbf{T}\mathbf{P}_2^{-1} = \mathbf{Y}(i,j) = (\alpha_i/\beta_j)\mathbf{T}(\pi_1(i), \pi_2(j))$, completing the proof. □

3.3 DC in the Cloud

See Algorithm 4 (Procedure DC_K-in-the-Cloud).

Algorithm 4. Procedure DC_K-in-the-Cloud

```
Input: $\mathbf{Y}$.
Output: $\mathbf{L}$ and $\mathbf{U}$.
1: On input the encrypted matrix $\mathbf{Y}$, the cloud then
   invokes any existing LU decomposition algorithm [21] to
   compute a lower triangular matrix $\mathbf{L}$ and an upper
   triangular matrix $\mathbf{U}$ such that $\mathbf{Y} = \mathbf{LU}$.
2: The cloud then sends $\mathbf{L}$ and $\mathbf{U}$ back to the client.
```

3.4 DC Decryption

See Algorithm 5 (Procedure DC-Decryption).

Algorithm 5. Procedure DC-Decryption

```
Input: L, U, and $K$.
Output: $D_X$.
 1: On receiving the returned matrices L and U from the cloud, the client computes $D_Y = \prod_{i=1}^{n+m}(\mathbf{L}_{ii}\mathbf{U}_{ii})$, where $D_Y$ is the unchecked determinant of matrix Y.
 2: The client further computes
    $$D_X = \frac{D_Y \times \mathrm{sgn}(\pi_2)\prod_{i=1}^{n+m}\beta_i}{\mathrm{sgn}(\pi_1)\prod_{i=1}^{n+m}\alpha_i \times \prod_{i=1}^{m} d_i},$$
    where $D_X$ is the unchecked determinant of matrix X.
```

3.5 Result Verification

Generally, handling result verification is not an easy task. However, this problem is well addressed by using the idea of Freivalds' algorithm [22], [23]. Technique details are elaborated in Algorithm 6 (Procedure Result-Verification). We defer the detailed analysis of it in Section 4.

Algorithm 6. Procedure Result-Verification

```
Input: The unchecked returned matrices L and U.
Output: Accepts $D_X$ as the correct result; or rejects it.
 1: If (L is not an $(n+m)\times(n+m)$ lower triangular matrix or U is not an $(n+m)\times(n+m)$ upper triangular matrix) then
 2:    Output "verification fails"; aborts.
 3: end if
 4: for $i = 1 : l$ do
 5:    The client generates an $(n+m)\times 1$ random 0/1 vector r.
 6:    The client computes $\mathbf{W} = \mathbf{L}\times(\mathbf{U}\mathbf{r}) - \mathbf{Y}\times\mathbf{r}$.
 7:    if ($\mathbf{W} \neq (0,\ldots,0)^{\mathrm{T}}$) then
 8:       Output "verification fails"; aborts.
 9:    end if
10: end for
11: The client accepts $D_X$ as the correct determinant of matrix X if the returned matrices pass the above check; otherwise, rejects it.
```

3.6 The Completed Protocol

We now present the completed protocol that contains five sub-algorithms (KeyGen, DCEnc, DCSolve, DCDec, ResultVerify) as follows:

- KeyGen(1^κ). On input a security parameter κ, the client invokes Algorithm 1 (Procedure Secret-Key-Generation) to get a secret key K: m, $\{d_1,\ldots,d_m\}$, $\{\alpha_1,\ldots,\alpha_{n+m}\}$, $\{\beta_1,\ldots,\beta_{n+m}\}$, $\mathbf{B}$, π_1, π_2, $\mathrm{sgn}(\pi_1)$, $\mathrm{sgn}(\pi_2)$.
- DCEnc$(\mathbf{X}; K)$. On input the original matrix **X** and the secret key K, the client invokes Algorithm 3 (Procedure DC-Encryption) to encrypt the original matrix **X** into an encrypted matrix **Y** to protect the input privacy.
- DCSolve$(\mathbf{Y})$. On input the encrypted matrix **Y**, the cloud invokes Algorithm 4 (Procedure DC$_K$-in-the-Cloud) to get two matrices **L** and **U**. Then, the cloud returns **L**, **U**, and an empty proof Γ to the client.
- DCDec$(\mathbf{L},\mathbf{U},K)$. On input the returned results **L** and **U**, and the secret key K, the client invokes Algorithm 5 (Procedure DC-Decryption) to get the unchecked determinant D_X of the original matrix **X**.
- ResultVerify$(\mathbf{L},\mathbf{U},\Gamma)$. On input the unchecked results **L**, **U**, and the empty proof Γ, the client invokes Algorithm 6 (Procedure Result-Verification) to check their correctness. If they pass the check, then the client accepts D_X as the correct determinant of the original matrix **X**; otherwise, just rejects it.

4 Correctness, Security, and Verifiability Analysis

4.1 Correctness Guarantee

Theorem 2. *The proposed protocol is correct.*

Proof. It suffices to show that if both the client and the cloud follow the protocol honestly, then $\det(\mathbf{X})$ is given by

$$\det(\mathbf{X}) = \frac{\det(\mathbf{Y}) \times \mathrm{sgn}(\pi_2)\prod_{i=1}^{n+m}\beta_i}{\mathrm{sgn}(\pi_1)\prod_{i=1}^{n+m}\alpha_i \times \prod_{i=1}^{m} d_i}, \tag{8}$$

where

$$\det(\mathbf{Y}) = \prod_{i=1}^{n+m}(\mathbf{L}_{ii}\mathbf{U}_{ii}). \tag{9}$$

If so, the decryption process in Algorithm 5 (Procedure DC-Decryption) will always yield the correct result.

Since $\mathbf{T} = [\begin{smallmatrix}\mathbf{X} & \mathbf{B}\\ \mathbf{0} & \mathbf{D}\end{smallmatrix}]$, it follows from Lemma 1 that

$$\det(\mathbf{X}) = \frac{\det(\mathbf{T})}{\det(\mathbf{D})} = \frac{\det(\mathbf{T})}{\prod_{i=1}^{m} d_i}. \tag{10}$$

According to $\mathbf{Y} = \mathbf{P}_1\mathbf{T}\mathbf{P}_2^{-1}$ and $\mathbf{Y} = \mathbf{L}\mathbf{U}$, one can obtain $\mathbf{T} = \mathbf{P}_1^{-1}\mathbf{L}\mathbf{U}\mathbf{P}_2$. This results in

$$\det(\mathbf{T}) = \det(\mathbf{P}_1)^{-1}\det(\mathbf{L})\det(\mathbf{U})\det(\mathbf{P}_2). \tag{11}$$

Remember, for triangle matrices, it holds that

$$\det(\mathbf{L}) = \prod_{i=1}^{n+m}\mathbf{L}_{ii}, \det(\mathbf{U}) = \prod_{i=1}^{n+m}\mathbf{U}_{ii}. \tag{12}$$

Combining (10), (11), (12) and results in Lemma 3, (8) is obtained. This implies that the decryption process will always yield the correct result and hence the proposed protocol is correct. □

4.2 Security Guarantee

4.2.1 Input Privacy

The proposed protocol can protect input privacy if the cloud cannot recover the original matrix **X** from the encrypted matrix $\mathbf{Y}$. The original matrix $\mathbf{X}$ is encrypted by the following three phases.

- *Phase 1*. The original matrix $\mathbf{X}$ serves as a building block of matrix $\mathbf{T}$, i.e., $\mathbf{T} = \left[\begin{smallmatrix}\mathbf{X} & \mathbf{B}\\ \mathbf{0} & \mathbf{D}\end{smallmatrix}\right]$.
- *Phase 2*. The position of each entry in the original matrix $\mathbf{T}$ is randomly rearranged under two random permutations, i.e., $\mathbf{U}(i,j) = \mathbf{T}(\pi_1(i), \pi_2(j))$.
- *Phase 3*. Each entry in matrix $\mathbf{U}$ is further masked by multiplying a factor, i.e., $\mathbf{Y}(i,j) = (\alpha_i/\beta_j)\mathbf{U}(i,j)$.

1) In Phase 1, if $m = 0$, then $\mathbf{T} = \mathbf{X}$. This will lead to $\mathbf{Y} = (\alpha_i/\beta_j)\mathbf{X}(\pi_1(i), \pi_2(j))$. Let N_X denote the number of zero entries in matrix $\mathbf{X}$. Then, by counting the number of zero entries in the encrypted matrix $\mathbf{Y}$, the cloud can obtain N_X. If $m \in \mathbb{Z}^+$, then $\mathbf{Y} = (\alpha_i/\beta_j)\mathbf{T}(\pi_1(i), \pi_2(j))$, where $\mathbf{T} = \left[\begin{smallmatrix}\mathbf{X} & \mathbf{B}\\ \mathbf{0} & \mathbf{D}\end{smallmatrix}\right]$. In this case, the cloud can obtain

$$N_X = N_T - N_B - nm - N_D = N_T - N_B - nm - (m^2 - m),$$

where $N_D = m^2 - m$ and N_B is randomly located in $\{0, \ldots, nm\}$. Suppose that n and m are revealed, then the cloud has a probability of correct guess of N_X to be $\frac{1}{nm}$. This probability is very small if nm is sufficiently large. In fact, the cloud only knows the information of $n + m$ from the encrypted matrix $\mathbf{Y}$, this reason further increases the difficulty to recover N_X. Therefore, N_X is well protected by using this block matrix technique.

2) In Phase 2, each entry in matrix $\mathbf{T}$ is rearranged under a random row permutation and a random column permutation. The key space consists of two random permutations π_1 and π_2. If the information of N_X remains secret, the cloud needs to guess $(m + n)!$ cases of permutations to recover $\mathbf{T}$ from matrix $\mathbf{U}$. What's more, the entries in matrix $\mathbf{B}$ and the diagonal entries in matrix $\mathbf{D}$ can be treated as "*noise entries*". In this Phase, these noise entries in the matrix $\mathbf{T}$ are diffused to the encrypted matrix $\mathbf{U}$ under two random permutations. Without the knowledge of the two permutations, these $(nm + m^2 - m)$ noise entries are indistinguishable from the entries in the original matrix $\mathbf{X}$. Intuitively, a larger choice of m will generate more noise entries in $\mathbf{B}$ and $\mathbf{D}$, which eventually results in a higher-security-level protocol.

3) In Phase 3, each entry in matrix $\mathbf{U}$ is further masked by a random scaling, the expected time of brute-force attack on the key space to guess $\{\alpha_1, \ldots, \alpha_{n+m}\}$ and $\{\beta_1, \ldots, \beta_{n+m}\}$ is $(|\mathcal{K}_\alpha|^{n+m}|\mathcal{K}_\beta|^{n+m})/2$. A choice of large key space $\mathcal{K}_\alpha$ and $\mathcal{K}_\beta$ will thwart this attack.

According to the above analysis, without the each-run-specific secret key, the cloud cannot recover $\mathbf{X}$ from $\mathbf{Y}$ by trivial means. The proposed protocol is believed to reach an applicable and strong enough security level in practice and hence input privacy is protected.

4.2.2 *Output Privacy*

The proposed protocol can protect output privacy if given the returned matrices $\mathbf{L}$ and $\mathbf{U}$, the cloud cannot recover the correct determinant $\det(\mathbf{X})$ of the original matrix $\mathbf{X}$. Given $\mathbf{L}$ and $\mathbf{U}$, the cloud can compute $\det(\mathbf{Y})$ using (9). Let $\gamma = \mathrm{sgn}(\pi_2)\prod_{i=1}^{n+m}\beta_i/(\mathrm{sgn}(\pi_1)\prod_{i=1}^{n+m}\alpha_i \times \prod_{i=1}^{m} d_i)$. It holds from (8) that $\det(\mathbf{X}) = \gamma \times \det(\mathbf{Y})$. Then, the cloud's task is to recover $\det(\mathbf{X})$ from $\det(\mathbf{Y})$. This attack is very hard based on the following two observations: 1) the range of γ can set to be very large via proper choices of key spaces $\mathcal{K}_d$, $\mathcal{K}_\alpha$, and $\mathcal{K}_\beta$; 2) a new key is generated in each run of the protocol, meaning that γ is different each time. As a consequence, without any prior knowledge of $\det(\mathbf{X})$ and the secret key K, given $\det(\mathbf{Y})$, the cloud still has a great uncertainty about $\det(\mathbf{X})$. In this way, the output privacy is protected.

4.3 Verifiability Guarantee

Theorem 3. *The proposed protocol satisfies robust cheating resistance.*

Proof. The correctness of the returned matrices $\mathbf{L}$ and $\mathbf{U}$ is checked form Steps 1 to 10 in Algorithm 6 (Procedure Result-Verification). The random check process from Steps 5 to 9 is repeated l times. We now define the following three parameters to facilitate our proof. Let $Prob_1$ be the probability of non-detection of false returned results (the results such that $\mathbf{Y} \neq \mathbf{LU}$) in one round of the random check process. Let $Prob_l$ denote the probability of non-detection of false returned results in whole l times random check processes. Moreover, we define $Prob_f$ to be the probability of *check failure*, i.e., the probability of non-detection of false returned results in Algorithm 6 (Procedure Result-Verification).

The proof consists of two steps. First, we show that the correct results from a faithful cloud server must be verified successfully by the client. Note that, if the cloud is faithful and the returned results are correct, then $\mathbf{L}$ must be an $(n+m) \times (n+m)$ lower triangular matrix and $\mathbf{U}$ must be an $(n+m) \times (n+m)$ upper triangular matrix, so the verification failure step (Step 2 in Algorithm 6 (Procedure Result-Verification)) will never be executed. Moreover, if the returned results are correct, we have $\mathbf{Y} = \mathbf{LU}$, so

$$\mathbf{W} = \mathbf{L} \times (\mathbf{Ur}) - \mathbf{Y} \times \mathbf{r} = (0, \ldots, 0)^{\mathrm{T}},$$

regardless of what vector $\mathbf{r}$ is. In such case, the other verification failure step (Step 8 in Algorithm 6 (Procedure Result-Verification)) will never be executed, either. This implies that the correct returned matrices $\mathbf{L}$ and $\mathbf{U}$ must be verified successfully by the client. Note also from Theorem 2 that if both $\mathbf{L}$ and $\mathbf{U}$ are correct, then in Algorithm 5 (Procedure DC-Decryption) the client will always get the correct determinant $\det(\mathbf{X})$ of the original matrix $\mathbf{X}$.

Next, we show that no false results from a cheating cloud server can pass the verification with a non-negligible probability. In other words, we attempt to prove that $Prob_f$ is a negligible quantity. We have the following two cases:

Case 1. It holds that $\mathbf{L}$ is not an $(n+m) \times (n+m)$ lower triangular matrix or $\mathbf{U}$ is not an $(n+m) \times (n+m)$ upper triangular matrix. In this case, the returned results must be false and the verification must fail, i.e., $Prob_f = 0$.

Case 2. It holds that $\mathbf{L}$ is an $(n+m) \times (n+m)$ lower triangular matrix and $\mathbf{U}$ is an $(n+m) \times (n+m)$ upper triangular matrix. Let

$$\mathbf{E} = \mathbf{LU} - \mathbf{Y}, \mathbf{W} = \mathbf{E} \times \mathbf{r} = (w_1, \ldots, w_{n+m})^{\mathrm{T}}.$$

If the cheating cloud returns false results (either **L** or **U**), then it can be deduced from Theorem 2 that D_X would not be the correct determinant of **X**. In this case, it holds that $\mathbf{LU} - \mathbf{Y} \neq \mathbf{0}$, so at least one entry of **E** is non-zero. Suppose that the entry $e_{ij} \neq 0$, by the definition of matrix-vector multiplication, we obtain

$$\begin{aligned} w_i &= \sum_{k=1}^{n+m} e_{ik} r_k = e_{i1} r_1 + \cdots + e_{ij} r_j + \cdots + e_{i(n+m)} r_{n+m} \\ &= e_{ij} r_j + y, \end{aligned} \tag{13}$$

where $y = \sum_{k=1}^{n+m} e_{ik} r_k - e_{ij} r_j$. By applying Total Probability Theorem [24], it holds that

$$\begin{aligned} \Pr[w_i = 0] = \Pr[w_i = 0 | y = 0] \Pr[y = 0] \\ + \Pr[w_i = 0 | y \neq 0] \Pr[y \neq 0] \end{aligned} \tag{14}$$

Note from (13) that

$$\begin{cases} \Pr[w_i = 0 | y = 0] = \Pr[r_j = 0] = 1/2, \\ \Pr[w_i = 0 | y \neq 0] \leq \Pr[r_j = 1] = 1/2. \end{cases} \tag{15}$$

Substituting (15) into (14) results in

$$\Pr[w_i = 0] \leq (1/2) \Pr[y = 0] + (1/2) \Pr[y \neq 0]. \tag{16}$$

Putting $\Pr[y \neq 0] = 1 - \Pr[y = 0]$ into (16) leads to

$$\Pr[w_i = 0] \leq 1/2.$$

Accordingly, it follows that

$$Prob_1 = \Pr[\mathbf{W} = (0, \ldots, 0)^{\mathrm{T}}] \leq \Pr[w_i = 0] \leq \frac{1}{2}.$$

Observe that the random check process is repeated l times, $Prob_l$ can be estimated by

$$Prob_l \leq Prob_1^l \leq \frac{1}{2^l},$$

Taken together, Cases 1 and 2 cover all possible cases, and so we have

$$Prob_f \leq Prob_l \leq \frac{1}{2^l}, \tag{17}$$

which demonstrates that $Prob_f$ is a negligible quantity in terms of l. The proof is hence completed. □

It can be deduced from the proof of Theorem 3 that the proposed protocol can handle result verification with check failure (non-detection of false results) probability at most 2^{-l}. The size of l is a tradeoff between efficiency and the probability of checking failure. In this paper, a choice of high cheating resistance should require l to be 40 bits (in this case $Prob_f \leq \frac{1}{2^{40}}$), for a fast check a reasonable choice of 10 bits is also acceptable (in this case $Prob_f \leq \frac{1}{2^{10}}$).

4.4 Further Discussions on Result Verification

Let us proceed to introduce the notion of Monte Carlo verification algorithm, which is formally defined below.

Definition 1 (Monte Carlo Verification Algorithm [23]). *The classification and the definition of Monte Carlo verification algorithm is summarized in Table 2. The detailed verbal description of case 1 is as follows: for a randomized verification algorithm Vrfy and any decrypted but unchecked result Res, if*

$$\begin{aligned} \Pr[\text{Vrfy accepts } Res \mid Res \text{ is correct}] = 1, \\ \Pr[\text{Vrfy accepts } Res \mid Res \text{ is false}] \leq \delta, \end{aligned}$$

then we define Vrfy as a true-biased Monte Carlo verification algorithm with one-sided error δ. The detailed verbal description of cases 2 and 3 can be analogously obtained.

Based on Definition 1 and the proof of Theorem 3, we immediately have the following theorem.

Theorem 4. *One round of random check process in Algorithm 6 (Procedure Result-Verification), i.e., from Steps 5 to 9, is a true-biased Monte Carlo verification algorithm with one-sided error* $\frac{1}{2}$.

For a Monte Carlo verification algorithm with one-sided error, the failure probability can be reduced (or the success probability amplified) by running the algorithm multiple times. Indeed, this mechanism has been exploited in Algorithm 6 (Procedure Result-Verification). According to the above analysis, cases 2 and 3 of Monte Carlo verification algorithms (see Table 2) can also be used in designing practical result verification algorithms for secure outsourcing. One may see in what follows that Monte Carlo verification algorithm offers superiority in designing efficient result verification algorithm, which is generally a difficult task in secure outsourcing.

5 Performance Evaluation

5.1 Theoretical Results

Client side overhead. The client side overhead is generated by running four sub-algorithms: KeyGen, DCEnc, DCDec, and ResultVerify. Since m is a constant, KeyGen takes time $O(n)$. Likewise, in DCEnc, applying (7) to efficiently compute **Y**, it only takes time $O(n^2)$. As to DCDec, the time needed is $O(n)$. The time consumed by ResultVerify is dominated by triangular matrices check and performing matrix-vector multiplication, which takes time $O(n^2)$ totally.

Cloud side overhead. For the cloud, its only computation overhead is generated by running DCSolve. The cloud can apply any existing LU decomposition algorithm. The most commonly used technique to perform matrix LU decomposition is using Gaussian elimination [21], which is of order $O(n^3)$. A significant result regarding to the lower bound of time for LU decomposition is given by [25]: if two matrices multiplication can be performed in time $O(n^\rho)$, for some $\rho > 2$, then the LU decomposition can be performed in time $O(n^\rho)$. This indicates that an $O(n^{2.373})$ algorithm exists based on the fastest matrix multiplication algorithm, i.e., Williams' algorithm [26]. Accordingly, suppose that this fastest LU decomposition algorithm is employed in the cloud side, then the time consumed by DCSolve is $O(n^{2.373})$.

Shown in Table 3 is a summarization of the theoretical results. The overall time cost is $O(n^2)$ for the client and $O(n^{2.373})$ for the cloud. From the perspective of efficiency, the proposed protocol is feasible due to the fact that there exists a gap between $O(n^2)$ and $O(n^{2.373})$. According to the

TABLE 2
Classification and Definition of Monte Carlo Verification Algorithm

Cases	**Satisfied conditions**	**Definitions**
case 1	$\Pr[\text{Vrfy accepts } Res \mid Res \text{ is correct}] = 1$, $\Pr[\text{Vrfy accepts } Res \mid Res \text{ is false}] \leq \delta$.	True-biased Monte Carlo verification algorithm with one-sided error δ
case 2	$\Pr[\text{Vrfy accepts } Res \mid Res \text{ is correct}] \geq \epsilon$, $\Pr[\text{Vrfy accepts } Res \mid Res \text{ is false}] = 0$.	False-biased Monte Carlo verification algorithm with one-sided error ϵ
case 3	$\Pr[\text{Vrfy accepts } Res \mid Res \text{ is correct}] \geq \epsilon$, $\Pr[\text{Vrfy accepts } Res \mid Res \text{ is false}] \leq \delta$.	Monte Carlo verification algorithm with two-sided errors (δ, ϵ)

TABLE 3
Theoretical Performance of the Proposed Protocol

Sides	Client					Cloud	
Sub-algorithms	KeyGen	DCEnc	DCDec	ResultVerify	Sending cost	DCSolve	Sending cost
Time complexity	$O(n)$	$O(n^2)$	$O(n)$	$O(n^2)$	Matrix $\mathbf{Y}$	$O(n^{2.373})$	Matrices $\mathbf{L}$ and $\mathbf{U}$

TABLE 4
Notations in Experiments

Notations	Means
t_{original}	the time for the client to compute the original DC locally
t_{cloud}	the time for the cloud to compute the outsourced DC_K
t_{client1}	the time for the client to generate the secret key and encrypt the original DC
t_{client2}	the time for the client to decrypt and verify the returned results
t_{client}	$t_{\text{client}} = t_{\text{client1}} + t_{\text{client2}}$
$\mathbf{Y}$	the matrix after encryption
$\mathbf{L}$, $\mathbf{U}$	the lower triangular matrix and the upper triangular matrix

asymptotic behavior of Big-O notation [27], we have that the computational overhead in the client side will be less than that in the cloud side for a sufficiently large n. The theoretical results indicate that the proposed protocol is able to allow the client to outsource DC to the cloud and gain substantial computation savings.

5.2 Experimental Results

Theoretical analysis of the protocol has shown that outsourcing indeed benefits the client. We proceed to implement the protocol to assess its practical efficiency in this subsection. Both client and cloud server computations in our experiments are conducted on the same workstation. If we implement the protocol for both client side and cloud side on the same workstation and measure their running time, then the ratio of time (see the definition of cloud efficiency and relative extra cost in the next paragraph) can reflect the asymmetric amount of computation performed in both sides. However, if we implement the protocol on two different workstations with one in client side and the other in cloud side, then the cloud efficiency and relative extra cost will be case-specific, depending on the asymmetric computing speed owned by the two different workstations. Consequently, one-workstation-based experiment is employed. Additionally, we ignore the communication latency between the client and the cloud for this application since the computation dominates the running time as shown in our experiments.

Our goal is to find the performance gain for the client by outsourcing. Thus, the main performance indicator is a ratio of the time that is needed if the computation is done locally over the time that is needed by the client's computation if outsourcing is chosen. With clear definition of parameters in Table 4, the performance gain of the client can be shown by $\frac{t_{\text{original}}}{t_{\text{client}}}$, which is referred to as *client speedup*. This value theoretically should be a considerable positive number greater than 1, which implies there is a considerable performance gain. Another metric is also taken into consideration, i.e., the *cloud efficiency*, using $\frac{t_{\text{original}}}{t_{\text{cloud}}}$. Ideally, the DC encryption should not increase the time to solve the original DC. It is desirable that the cloud efficiency is close to 1. What's more, we would like to see a measurement of the extra cost, which is defined as the amount of time cost at both client and the cloud deducted by the time cost incurred by the client only, i.e., $t_{\text{cloud}} + t_{\text{client}} - t_{\text{original}}$. With this notion, we define the *relative extra cost* (REC) as

$$\frac{t_{\text{cloud}} + t_{\text{client}} - t_{\text{original}}}{t_{\text{original}}} = \frac{t_{\text{client}} + t_{\text{cloud}}}{t_{\text{original}}} - 1.$$

It is desirable for REC to stay close to 0, which implys that there are not much extra work incurred by employing the proposed outsourcing protocol.

Both client and cloud server computations in our experiments are conducted on the same workstation, which is equipped with an Intel Xeon CPU E5620 and 32 GB RAM. We implement the proposed protocol in Matlab and utilize the LAPACK [28] package to perform determinant computation and LU decomposition. Due to the fact that loops are extremely slow in matlab, the interface provided by MEXfile is employed to invoke code in C language whenever running loops. All of matrix instances in experiments are generated to be full rank with each entry randomly located in $(0, 1)$. We set $\mathcal{K}_d = (0, 1), \mathcal{K}_\alpha = \mathcal{K}_\beta = \{1, \ldots, n+m\}$ and design two groups of experiments as described below.

Experiment 1. We first fix $l = 20$ and conduct three experiments with $m = 100$, $m = 200$, and $m = 400$, which correspond to security priority, tradeoff, and efficiency priority cases.

Experiment 2. We then fix $m = 200$ and conduct three experiments with $l = 10$, $l = 20$, and $l = 40$, which correspond to client speedup priority, tradeoff, and cheating resistance priority cases.

The main performance is shown in Tables 5 and 6. It can be observed that client speedup is monotonically increasing with matrix dimension n. Outsourcing DC is able to gain more than 6 times client speedup if n is sufficiently large. Besides, it is shown from Tables 5 and 6 that the cloud efficiency stays close to 1, which is very satisfactory. Finally, it also can be seen that REC is monotonically decreasing with matrix dimension n. The comparisons of client speedup, cloud efficiency, and REC as a function of matrix dimension n are depicted in Fig. 3.

As evidenced by Tables 5 and 6, and Fig. 3, the proposed protocol offers us two levels of tradeoff to choose in practical applications. 1) *The First Tradeoff.* The choice of m is a tradeoff between security and efficiency (including client speedup, cloud efficiency, and REC). It can be found in Figs. 3a, 3b, and 3c that a larger choice of m indicates a more secure yet less efficient protocol. 2) *The Second Tradeoff.* The choice of l is a tradeoff between cheating resistance and efficiency (including client speedup and REC). Figs. 3d and 3f demonstrates that a larger choice of l leads to a degradation of client speedup and an increase of REC. It is worth noting from Fig. 3e that the choice of l does not affect the cloud efficiency.

Remarkably, the experimental performance really depends on matrix dimension, code compile platform, and the selected algorithm for LU decomposition in the cloud side. If the cloud exploits other faster LU decomposition algorithms, then client speedup will decrease to some extent. However, as long as n goes sufficiently large, the substantial computation savings can always be anticipated by the client due to the clear existence of gap between $O(n^2)$ and $O(n^{2.373})$.

6 Related Work

Secure outsourcing, since its proposal, has stimulated considerable research efforts both from theoretical cryptographers and security engineers. With the advent of cloud and mobile computing age, the theoretical cryptographers' interest in secure outsourcing is persistently increasing, especially after Gentry's first FHE scheme [29] by using an ideal lattice. They often focus on designing a generic protocol that covers all problems, e.g., [30], [31]. The generic protocol always involves in applying a FHE scheme, which is a cryptographic primitive that seems to be far from practical. Hence, the generic protocol is currently quite complicated and inefficient. As to security engineers, they often identify some specific problems and design different techniques to mask the original problem to protect input/output privacy. Their protocol always lack formal security treatment and do not handle the important case of result verification, but

TABLE 5
Performance with Fixed l (Time in Seconds)

Benchmark		Original DC	Encrypted DC_K				Client Speedup	Cloud Efficiency	Relative Extra Cost
No.	dimension n	t_{original}	t_{cloud}	t_{client1}	t_{client2}	t_{client}	$\frac{t_{\text{original}}}{t_{\text{client}}}$	$\frac{t_{\text{original}}}{t_{\text{cloud}}}$	$\frac{t_{\text{client}}+t_{\text{cloud}}}{t_{\text{original}}} - 1$
1	5000	6.594	7.314	1.505	1.360	2.865	2.302×	0.9016	0.5435
2	10000	45.82	49.69	6.242	6.038	12.28	3.731×	0.9221	0.3525
3	15000	145.7	154.0	16.69	11.80	28.49	5.115×	0.9461	0.2525
4	20000	335.4	350.5	34.88	22.84	57.72	5.810×	0.9568	0.2173

I. High Security and Low Efficiency Case: $m = 100$ and $l = 20$ ($Prob_f \leq \frac{1}{2^{20}}$)

Benchmark		Original DC	Encrypted DC_K				Client Speedup	Cloud Efficiency	Relative Extra Cost
No.	dimension n	t_{original}	t_{cloud}	t_{client1}	t_{client2}	t_{client}	$\frac{t_{\text{original}}}{t_{\text{client}}}$	$\frac{t_{\text{original}}}{t_{\text{cloud}}}$	$\frac{t_{\text{client}}+t_{\text{cloud}}}{t_{\text{original}}} - 1$
1	5000	6.693	8.014	1.602	1.560	3.162	2.117×	0.8351	0.6698
2	10000	45.46	50.35	6.508	6.176	12.68	3.585×	0.9028	0.3866
3	15000	146.6	158.0	17.61	13.44	31.05	4.722×	0.9281	0.2892
4	20000	336.8	357.9	37.01	23.46	60.47	5.570×	0.9409	0.2423

II. Tradeoff between Security and Efficiency Case: $m = 200$ and $l = 20$ ($Prob_f \leq \frac{1}{2^{20}}$)

Benchmark		Original DC	Encrypted DC_K				Client Speedup	Cloud Efficiency	Relative Extra Cost
No.	dimension n	t_{original}	t_{cloud}	t_{client1}	t_{client2}	t_{client}	$\frac{t_{\text{original}}}{t_{\text{client}}}$	$\frac{t_{\text{original}}}{t_{\text{cloud}}}$	$\frac{t_{\text{client}}+t_{\text{cloud}}}{t_{\text{original}}} - 1$
1	5000	6.726	9.183	1.796	1.775	3.572	1.883×	0.7325	0.8963
2	10000	45.64	53.99	6.932	6.246	13.18	3.464×	0.8453	0.4717
3	15000	146.6	163.7	19.44	13.18	32.62	4.495×	0.8954	0.3393
4	20000	336.0	366.6	38.89	23.11	62.00	5.419×	0.9165	0.2756

III. Low Security and High Efficiency Case: $m = 400$ and $l = 20$ ($Prob_f \leq \frac{1}{2^{20}}$)

TABLE 6
Performance with Fixed m (Time in Seconds)

Benchmark		Original DC	Encrypted DC_K				Client Speedup	Cloud Efficiency	Relative Extra Cost
No.	dimension n	t_{original}	t_{cloud}	t_{client1}	t_{client2}	t_{client}	$\frac{t_{\text{original}}}{t_{\text{client}}}$	$\frac{t_{\text{original}}}{t_{\text{cloud}}}$	$\frac{t_{\text{client}}+t_{\text{cloud}}}{t_{\text{original}}}-1$
1	5000	6.560	15.77	1.517	0.7453	2.262	2.900×	0.8318	0.5478
2	10000	46.01	51.00	6.419	2.928	9.347	4.922×	0.9021	0.3117
3	15000	146.5	157.8	18.27	6.305	24.58	5.962×	0.9288	0.2444
4	20000	336.7	355.9	37.23	11.93	49.16	6.850×	0.9462	0.2028

I. High Client Speedup and Low Cheating Resistance Case: $m=200$ and $l=10$ ($Prob_f \leq \frac{1}{2^{10}}$)

Benchmark		Original DC	Encrypted DC_K				Client Speedup	Cloud Efficiency	Relative Extra Cost
No.	dimension n	t_{original}	t_{cloud}	t_{client1}	t_{client2}	t_{client}	$\frac{t_{\text{original}}}{t_{\text{client}}}$	$\frac{t_{\text{original}}}{t_{\text{cloud}}}$	$\frac{t_{\text{client}}+t_{\text{cloud}}}{t_{\text{original}}}-1$
1	5000	6.652	7.819	1.640	1.479	3.119	2.133×	0.8508	0.6442
2	10000	45.59	50.08	6.483	5.986	12.47	3.656×	0.9104	0.3719
3	15000	145.2	157.0	17.59	12.11	29.70	4.890×	0.9248	0.2858
4	20000	339.7	356.5	38.90	22.78	61.68	5.507×	0.9528	0.2311

II. Tradeoff between Client Speedup and Cheating Resistance Case: $m = 200$ and $l = 20$ ($Prob_f \leq \frac{1}{2^{20}}$)

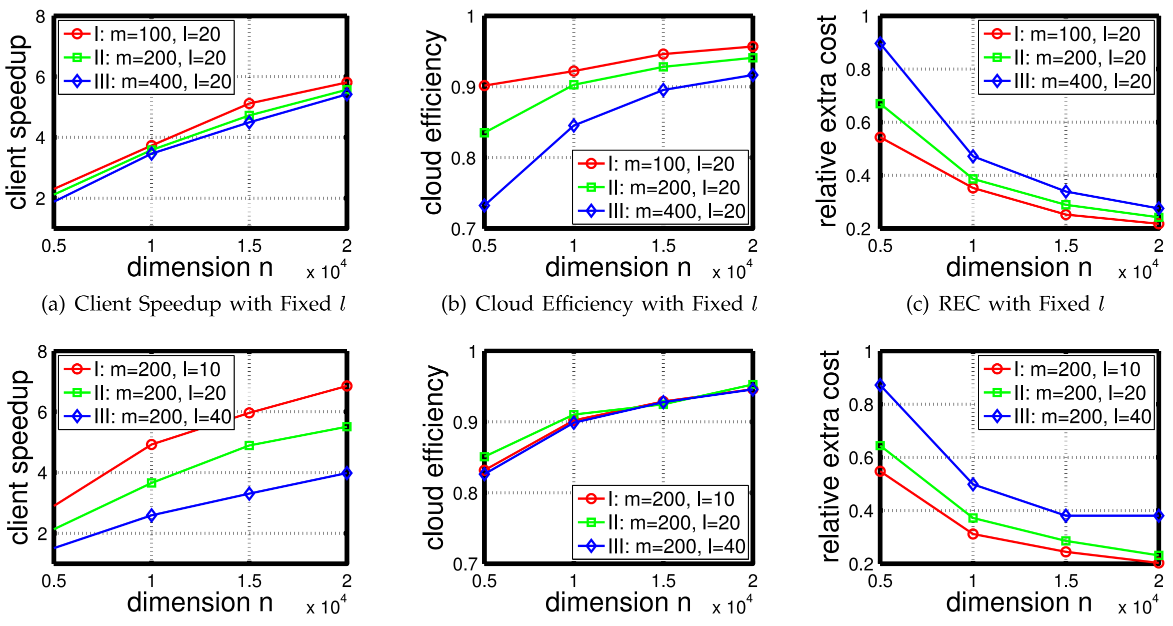
Benchmark		Original DC	Encrypted DC_K				Client Speedup	Cloud Efficiency	Relative Extra Cost
No.	dimension n	t_{original}	t_{cloud}	t_{client1}	t_{client2}	t_{client}	$\frac{t_{\text{original}}}{t_{\text{client}}}$	$\frac{t_{\text{original}}}{t_{\text{cloud}}}$	$\frac{t_{\text{client}}+t_{\text{cloud}}}{t_{\text{original}}}-1$
1	5000	6.651	8.049	1.639	2.764	4.403	1.511×	0.8263	0.8720
2	10000	45.67	50.82	6.561	11.06	17.63	2.591×	0.8988	0.4985
3	15000	146.93	158.4	17.03	27.42	44.45	3.305×	0.9278	0.3804
4	20000	337.9	357.3	35.93	48.92	84.86	3.982×	0.9458	0.3804

III. Low Client Speedup and High Cheating Resistance Case: $m = 200$ and $l = 40$ ($Prob_f \leq \frac{1}{2^{40}}$)

these protocols are always quite efficient and can be deployed immediately.

6.1 Works for Specific Applications

Over the past few decades, many protocols have been designed for secure outsourcing of some specific applications. For example, Atallah et al. [20] propose a number of protocols for secure outsourcing scientific computations, such as solving a linear system of equations, sorting, etc. They employed a lot of problem transformation techniques to construct the protocols, but the common drawbacks of their protocols are two-fold: they lack detailed efficiency analysis and evaluation, and they do not tackle the issue of result verification. Until recently, two secure matrix multiplication outsourcing protocols were introduced in [32] and [33]. The former is built upon the assumptions of two



(a) Client Speedup with Fixed l (b) Cloud Efficiency with Fixed l (c) REC with Fixed l

(d) Client Speedup with Fixed m (e) Cloud Efficiency with Fixed m (f) REC with Fixed m

Fig. 3. Comparisons of Client Speedup, Cloud Efficiency, and REC.

non-colluding servers, making it vulnerable to colluding attacks. While the later achieves provable security using Shamir's secret sharing [34] technique. But this theoretically elegant protocol still suffers from large amount of communication overhead. Several schemes for ranked keyword searching over outsourced cloud data are developed in [35] by virtue of harnessing the order-preserving symmetric encryption technique in [36]. Notably, Wang et al. investigate the issue of privacy-assured outsourcing of image reconstruction service (OIRS) in cloud [4]. The proposed OIRS protocol can additionally support extensible service interfaces and even performance speedup via hardware built-in design. After Gentry's breakthrough work on FHE scheme, the research direction is currently shifting to design secure outsourcing protocol in the malicious cloud model rather than in the fully trusted cloud model. Hence, it is essential to handle result verfication. Following this trend, several protocols that can handle result verification are proposed, among which there are the secure outsourcing of linear programming [37], the secure outsourcing of linear equations [38], the secure outsourcing of convex optimization [39], and the secure outsourcing of matrix inversion and multiplication [40], [41], etc. Recently, the works in [37] and [38] are further improved in [42] by employing some special linear transformations and a pseudorandom number generator. Our system model and framework are inherited from these works.

6.2 Functionally Related Work

There are three kinds of existing work that are conceptually and functionally related to secure outsourcing. The first one is secure multi-parity computation (SMC), initially introduced by Yao [43]. The goal of SMC to create methods that enable parties to jointly compute a function over their inputs, while at the same time keeping these inputs private. For example, two millionaires can compute which one is richer, but without revealing their net worth. Generally, the parties in SMC is symmetry assigned with the same computational tasks, whereas in outsourcing systems, the parties are resource-asymmetry, i.e., a weak client and a powerful cloud server. Because SMC does not consider the asymmetry between the resources possessed by cloud and client, it cannot be applied to secure outsourcing directly. The second one is about delegating computation and cheating detection, e.g., [44]. Yet, the traditional work on cheating detection allows the server to access the original data, which is prohibited in the proposed secure outsourcing paradigm. The third one is server-aided computations, such as [45], [46]. One limitation of these protocols is that they are mainly concerned with outsourcing of cryptographic computations like signature and modular exponentiation. The other limitation is that these protocols do not handle result verification.

7 Conclusions and Outlook

In this paper, we have designed a state-of-the-practice protocol for outsourcing of DC to a malicious cloud. It is shown that the proposed protocol simultaneously fulfills the goals of correctness, security (input/output privacy), robust cheating resistance, and high-efficiency. With the advent of large-scale data and cloud computing era, there is an increasing need for a well-integrated scientific computations outsourcing software system, which should be able to provide secure outsourcing services for all kinds of commonly used scientific computations. We conceive that such integrated software system owns interface in the client side like Matlab, whereas it moves the heavy computation to the cloud server. We have designed a state-of-the-practice protocol for outsourcing of DC in this paper, a step forward to build such integrated software system. In order to accelerate the birth of such service-oriented software system, one future work is to identify new commonly used scientific computations and then designing protocols to solve them. It is hoped that the proposed protocol can shed light in designing other novel secure outsourcing protocols, and inspire powerful companies and working groups to finish the programming of the envisioned integrated software system. It is believed that such software system can be profitable by means of providing large-scale scientific computation services for so many potential clients.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61170249, in part by the Research Fund of Preferential Development Domain for the Doctoral Program of Ministry of Education of China under Grant 201101911130005, in part by the Talents of Science and Technology Promote Plan, Chongqing Science & Technology Commission, and in part by the Program for Changjiang Scholars. This work was made possible by NPRP Grant 4-1162-1-181 from the Qatar National Research Fund (a member of the Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

[1] L.-J. L. Zhang, "Editorial: Big services era: Global trends of cloud computing and big data," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 0467–468, fouth quarter 2012.

[2] K. P. Joshi, Y. Yesha, and T. Finin, "Automating cloud services life cycle through semantic technologies," *IEEE Trans. Serv. Comput.*, vol. 7, no. 1, pp. 109–122, Jan.-Mar. 2014.

[3] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," *IEEE Trans. Serv. Comput.*, vol. 5, no. 2, pp. 220–232, Apr.-Jun. 2012.

[4] C. Wang, B. Zhang, K. Ren, and J. Wang, "Privacy-assured outsourcing of image reconstruction service in cloud," *IEEE Trans. Emerg. Topics in Comput.*, vol. 1, no. 1, pp. 166–177, Jun. 2013.

[5] I. Paik, W. Chen, and M. N. Huhns, "A scalable architecture for automatic service composition," *IEEE Trans. Serv. Comput.*, vol. 7, no. 1, pp. 82–95, Jan.-Mar. 2014.

[6] K.-W. Park, J. Han, J. Chung, and K. H. Park, "THEMIS: A mutually verifiable billing system for the cloud computing environment," *IEEE Trans. Serv. Comput.*, vol. 6, no. 3, pp. 300–313, Jul.-Sep. 2014.

[7] Y. Zhou, Y. Zhang, H. Liu, N. Xiong, and A. V. Vasilakos, "A bare-metal and asymmetric partitioning approach to client virtualization," *IEEE Trans. Serv. Comput.*, vol. 7, no. 1, pp. 40–53, Jan.-Mar. 2014.

[8] G. Brunette and R. Mogull, "Security guidance for critical areas of focus in cloud computing v2. 1," *Cloud Security Alliance*, pp. 1–76, 2009.

[9] X. Zhang, L. T. Yang, C. Liu, and J. Chen, "Oruta: Privacy-preserving public auditing for shared data in the cloud," *IEEE Trans. Serv. Comput.*, vol. 2, no. 1, pp. 43–56, Jan.-Mar. 2014.

[10] B. Peng, "The determinant: A means to calculate volume," *Recall*, vol. 21, pp. 1–6, 2007.

[11] C. Meyer, *Matrix Analysis and Applied Linear Algebra Book and Solutions Manual*, vol. 2, Philadelphia, PA, USA: SIAM, 2000.

[12] R. Vein, P. Dale, R. Vein, and P. Dale, *Determinants and Their Applications in Mathematical Physics*, vol. 1, New York, NY, USA: Springer, 1999.

[13] C. K. Chui and M.-J. Lai, "Vandermonde determinant and lagrange interpolation in rs," in *Nonlinear and Convex Analysis*, vol. 107, Boca Raton, FL, USA: CRC Press, 1987, pp. 23–35.

[14] S. Biswas, K. Datta, R. Saxena, P. Srivastava, and V. Varma, "The hill determinant: An application to the anharmonic oscillator," *Phys. Rev. D*, vol. 4, no. 12, pp. 3617–3620, 1971.

[15] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *J. Privacy Confidentiality*, vol. 1, no. 1, pp. 59–98, 2009.

[16] R. C. Lyndon, P. E. Schupp, R. Lyndon, and P. Schupp, *Combinatorial Group Theory*, vol. 177, Berlin, Germany: Springer-Verlag, 1977.

[17] G. B. Arfken, H. J. Weber, and F. E. Harris, *Mathematical Methods for Physicists: A Comprehensive Guide*. New York, NY, USA: Academic, 2011.

[18] R. Durstenfeld, "Algorithm 235: Random permutation," *Commun. ACM*, vol. 7, no. 7, pp. 420–420, 1964.

[19] D. E. Knuth, *The Art of Computer Programming*. Reading, MA, USA: Addison-Wesley, 2006.

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

[21] C. F. Gerald and P. O. Wheatley, *Numerical Analysis*. Reading, MA, USA: Addison-Wesley, 2003.

[22] R. Freivalds, "Probabilistic machines can use less running time," *Inf. Process.*, vol. 77, pp. 839–842, 1977.

[23] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995.

[24] R. Durrett, *Probability: Theory and Examples*, vol. 3, Cambridge, U.K.: Cambridge Univ. Press, 2010.

[25] J. R. Bunch and J. E. Hopcroft, "Triangular factorization and inversion by fast matrixmatrix multiplication," *Math. Comput.*, vol. 28, no. 125, pp. 231–236, 1974.

[26] Y. Chen and P. Nguyen, "Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers," in *Proc. 31st Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2012, pp. 502–519.

[27] C. H. Papadimitriou, *Computational Complexity*. Hoboken, NJ, USA: Wiley, 2003.

[28] E. Anderson, *LAPACK Users' Guide*, vol. 9, Philadelphia, PA, USA: SIAM, 1999.

[29] C. Gentry, "*A fully homomorphic encryption scheme*," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.

[30] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. 30th Annu. Conf. Adv. Cryptology*, 2010, pp. 465–482.

[31] K. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Proc. 30th Annu. Conf. Adv. Cryptology*, 2010, pp. 483–501.

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

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

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

[35] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1467–1479, Aug. 2012.

[36] A. Boldyreva, N. Chenette, Y. Lee, and A. Oneill,, "Order-preserving symmetric encryption," in *Proc. 28th Annu. Int. Conf. Adv. Cryptology: Theory Appl. Cryptograph. Techn.*, 2009, pp. 224–241.

[37] C. Wang, K. Ren, and J. Wang, "Secure and practical outsourcing of linear programming in cloud computing," in *Proc. IEEE Conf. Comput. Commun.*, 2011, pp. 820–828.

[38] C. Wang, K. Ren, J. Wang, and Q. Wang, "Harnessing the cloud for securely outsourcing large-scale systems of linear equations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1172–1181, Jun. 2012.

[39] Z. Xu, C. Wang, Q. Wang, K. Ren, and L. Wang, "Proof-carrying cloud computation: The case of convex optimization," in *Proc. IEEE Conf. Comput. Commun.*, 2013, pp. 610–614.

[40] 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, pp. 78–87, Jan.-Jun. 2013.

[41] 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, 2014.

[42] F. Chen, T. Xiang, and Y. Yang, "Privacy-preserving and verifiable protocols for scientific computation outsourcing to the cloud," *J. Parallel Distrib. Comput.*, vol. 74, no. 3, pp. 2141–2151, 2014.

[43] A. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, 1982, pp. 160–164.

[44] S. Goldwasser, Y. Kalai, and G. Rothblum, "Delegating computation: Interactive proofs for muggles," in *Proc. 40th Annu. ACM Symp. Theory Comput.*, 2008, pp. 113–122.

[45] S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations," in *Proc. 2nd Int. Conf. Theory Cryptography*, 2005, pp. 264–282, 2005.

[46] S. Kawamura and A. Shimbo, "Fast server-aided secret computation protocols for modular exponentiation," *IEEE J. Sel. Areas Commun.*, vol. 11, no. 5, pp. 778–784, Jun. 1993.

Xinyu Lei received the BS degree in computing science from Chongqing University, China, in 2010, where he is currently toward the PhD degree in computer science. He was once a visiting scholar at Texas A&M University at Qatar, Doha. His research interests include information security and algorithms.

Xiaofeng Liao received the BS and MS degrees in mathematics from Sichuan University, Chengdu, China, in 1986 and 1992, respectively, and the PhD degree in circuits and systems from the University of Electronic Science and Technology of China in 1997. His current research interests include neural networks, nonlinear dynamical systems, bifurcation and chaos, and cryptography. He is a senior member of the IEEE.

Tingwen Huang received the BS degree from Southwest Normal University (now Southwest University), China, in 1990, the MS degree from Sichuan University, China, in 1993, and the PhD degree from Texas A&M University, College Station, in 2002. He was the president of the Asia Pacific Neural Networks Assembly in 2012. He is currently an associate editor for the *IEEE Transactions on Neural Networks and Learning Systems*.

Huaqing Li received the BS degree in information and computation science from the Chongqing University of Posts and Telecommunications, China, in 2009. He is currently working toward the PhD degree in computer science at Chongqing University. His research interests include nonlinear dynamical systems, bifurcation and chaos, and consensus of multiagent systems.