Information Sciences 280 (2014) 205–217

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud

Xinyu Lei [a,*], Xiaofeng Liao [a], Tingwen Huang [b], Feno Heriniaina [a]

[a] The State Key Lab. of Power Transmission Equipment & System Security and New Technology, College of Computer Science, Chongqing University, Chongqing, China
[b] Texas A&M University at Qatar, P.O. Box 23874, Doha, Qatar

ARTICLE INFO

Article history:
Received 22 June 2013
Received in revised form 6 May 2014
Accepted 10 May 2014
Available online 16 May 2014

Keywords:
Cloud computing
Matrix multiplication
Secure outsourcing
Monte Carlo verification

ABSTRACT

Computation outsourcing to the cloud has become a popular application in the age of cloud computing. This computing paradigm brings in some new security concerns and challenges, such as input/output privacy and result verifiability. Given that matrix multiplication computation (MMC) is a ubiquitous scientific and engineering computational task, we are motivated to design a protocol to enable secure, robust cheating resistant, and efficient outsourcing of MMC to a malicious cloud in this paper. The main idea to protect the privacy is employing some transformations on the original MMC problem to get an encrypted MMC problem which is sent to the cloud; and then transforming the result returned from the cloud to get the correct result to the original MMC problem. Next, a randomized Monte Carlo verification algorithm with one-sided error is introduced to successfully handle result verification. We analytically show that the proposed protocol is correct, secure, and robust cheating resistant. Extensive theoretical analysis and experimental evaluation also show its high-efficiency and immediate practicability. Finally, comparisons between the proposed protocol and the previous protocols are given to demonstrate the improvements of the proposed protocol.

1. Introduction

With the emergence of the cloud computing paradigm in scientific and business applications, it has become increasingly important to provide service-oriented computing in third-party data management settings [24,23]. Cloud computing is capable of providing massive computing resources to clients as services while hiding implementation details from clients [1,32]. With this paradigm, the resource-constrained clients can off-load their intensive computational tasks to clouds, which are equipped with massive computational resources. 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 cloud servers.

1.1. Challenges

Although it is quite promising, outsourcing computational problem to the commercial public inevitably brings in new security concerns and challenges [6,21,22,35]. The first challenge is the client's input/output data privacy. The outsourced

* Corresponding author. Tel.: +86 13658371220.
E-mail address: xy-lei@qq.com (X. Lei).

http://dx.doi.org/10.1016/j.ins.2014.05.014
0020-0255/

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 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

Matrix multiplication computation (MMC) is a basic computational problem in scientific and engineering fields and has a number of applications. This can be well illustrated by the following examples. MMC is frequently used in statistics theory. Take a typical linear regression model $\mathbf{y} = \mathbf{X}\beta$ as an example, the least squared error method yields an solution for β by computing $\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ [29]. Besides, MMC plays an important role in linear algebra and matrix theory [26]. For instance, the linear discrete dynamical systems are best studied in a matrix formulation $\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n$, where the solution is $\mathbf{x}_n = \mathbf{A}^n\mathbf{x}_0$. The computation of $\mathbf{A}^n$ allows us to obtain this solution. Moreover, MMC is well rooted in many other scientific and engineering fields including image encryption [31,38], 3D graphics simulations [13], discriminant analysis [36,27], sliding mode analysis [16,4], to just list a few. In short, MMC is widely needed for a variety of potential clients. When the restricted computational resources are possessed by these clients and MMC deals with large matrices, an economical solution is to outsource MMC 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 MMC is preferred. Consequently, we are motivated to design a protocol that enables clients to securely, verifiably, and efficiently outsource MMC to a cloud.

1.3. Contributions

We regard our main contributions as fourfold:

1. We identify a common scientific and engineering computational task, i.e., matrix multiplication computation outsourcing, and then design a protocol to fulfill it.
2. We show that the proposed protocol can simultaneously achieve goals of correctness, security, robust cheating resistance, and efficiency.
3. By introducing Monte Carlo verification algorithm, the problem of result verification is well addressed. Additionally, the superiority of Monte Carlo verification algorithm in designing inexpensive result verification algorithm for secure outsourcing is well demonstrated.
4. We show by theoretical analysis and experimental evaluation that the proposed protocol is highly efficient, and therefore, it can be deployed in practical applications immediately.

1.4. Organization

The remainder of this paper proceeds as follows. Section 2 introduces some essential preliminaries. In Section 3, we describe the proposed 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 MMC outsourcing system model, as illustrated in Fig. 1. A client with low computational power intends to outsource the multiplication computation of matrices $\mathbf{X}$ and $\mathbf{Y}$, denoted as $\Phi = (\mathbf{X}, \mathbf{Y})$, to a cloud service provider, who has massive computational power and special softwares. In order to protect input privacy, the client encrypts the original MMC problem Φ using a secret key K to get a new computational problem, written as Φ_K. Later, the encrypted

Φ_K is given to the cloud for a result. Once the cloud receives Φ_K, the computation is carried out with softwares; then the cloud sends back the result to Φ_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 the returned result using the secret key K to get the result to the original MMC problem Φ. Meanwhile, the client checks whether this result is correct: if yes, accepts it; if no, 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 [20]. 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 [20]. 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. The proposed protocol should be able to resist such a malicious cloud.

2.1.3. Design goals

We identify the following goals that the outsourcing protocol should satisfy.

- **Correctness.** If both the client and the cloud follow the protocol honestly, the original MMC Φ can be indeed fulfilled by the cloud and the client gets a correct result to Φ.
- **Security.** The protocol can protect the privacy of the client's data. On one hand, given the encrypted Φ_K, the cloud cannot get meaningful knowledge of the client's original input data Φ, which is referred to as *input privacy*. On the other hand, the correct result to the original MMC problem Φ is also hidden from the cloud, and this is called as *output privacy*.
- **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.
- **Efficiency.** The local computation done by the client should be substantially less than the computation of the original MMC Φ on its own. In addition, the amount of computation on computing the encrypted Φ_K should be as close as possible to that on computing the original Φ.

2.1.4. Framework

Syntactically, a secure MMC outsourcing protocol should contain five sub-algorithms: (1) the algorithm for key generation KeyGen, (2) the algorithm for MMC encryption MMCEnc, (3) the algorithm for solving MMC_K problem MMCSolve, (4) the algorithm for MMC decryption MMCDec, 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 processes 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 provide one-time-pad type of flexibility. That is to say, KeyGen will be run every time for a new outsourced MMC instance to enhance security. Once we have this framework, we just need to work out the details of these four sub-algorithms, which will be shown in Section 3.

2.2. Mathematical background

Permutation function is well studied in group theory and combinatorics. In Cauchy's two-line notation, 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. Then the permutation function can be written as:

$$\begin{pmatrix} 1 & \dots & m \\ p_1 & \dots & p_m \end{pmatrix}. \tag{1}$$

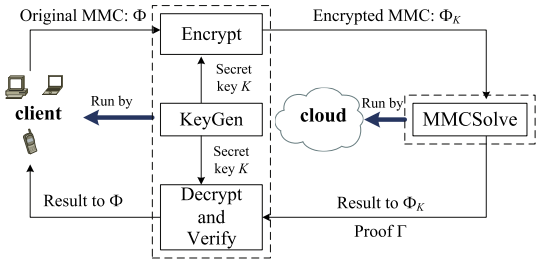

Fig. 1. Secure MMC outsourcing system model.

We use a permutation function $\pi(i) = p_i$, where $i = 1, \ldots, m$, to denote (1). Let π^{-1} denote the inverse function of π. We denote by $\pi \leftarrow \mathrm{RandP}(1, \ldots, m)$ the process of generating a random permutation π of preimage as integers $1, \ldots, m$. We write $\{k_1, \ldots, k_m\} \leftarrow \mathcal{K}$ to denote $\{k_1, \ldots, k_m\}$ are all chosen uniformly at random from the key space $\mathcal{K}$. The Kronecker delta function $\delta_{x,y}$ is defined as

$$\delta_{x,y} = \begin{cases} 1, & x = y, \\ 0, & x \neq y. \end{cases} \tag{2}$$

Let $\mathbf{X}(i,j), x_{i,j}$, or x_{ij} denote the entry in ith row and jth column in matrix $\mathbf{X}$, where i and j are indexed from 1 to n.

3. Protocol construction

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

3.1. Secret key generation

Consider a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ and a matrix $\mathbf{Y} \in \mathbb{R}^{n \times s}$, where necessarily the number of columns in $\mathbf{X}$ equals to the number of rows in $\mathbf{Y}$. The resource-constrained client wants to securely outsource the computation of $\mathbf{XY}$ to the cloud. The protocol starts by invoking Procedure Secret-Key-Generation to set up a secret key K.

Algorithm 1. Procedure Secret-Key-Generation

Input: A security parameter λ.
Output: Secret key $K : \{\alpha_1, \ldots, \alpha_m\}, \{\beta_1, \ldots, \beta_n\}, \{\gamma_1, \ldots, \gamma_s\}, \pi_1, \pi_2, \pi_3$.
1: On input a security parameter λ, which specifies three key spaces $\mathcal{K}_\alpha, \mathcal{K}_\beta$, and $\mathcal{K}_\gamma$, the client picks three sets of random numbers: $\{\alpha_1, \ldots, \alpha_m\} \leftarrow \mathcal{K}_\alpha, \{\beta_1, \ldots, \beta_n\} \leftarrow \mathcal{K}_\beta$, and $\{\gamma_1, \ldots, \gamma_s\} \leftarrow \mathcal{K}_\gamma$, where $0 \notin \mathcal{K}_\alpha \cup \mathcal{K}_\beta \cup \mathcal{K}_\gamma$.
2: The client invokes Algorithm 2 to generate three random permutations: $\pi_1 \leftarrow \mathrm{RandP}(1, \ldots, m), \pi_2 \leftarrow \mathrm{RandP}(1, \ldots, n)$, and $\pi_3 \leftarrow \mathrm{RandP}(1, \ldots, s)$.

Algorithm 2. Random Permutation Generation

Output: a permutation π of integers $1, \ldots, m$.
1: Set $\pi = \mathrm{I}_m$. (identical permutation)
2: **for** $i = m$ down to 2
3: Set j to be a random integer with $1 \leqslant j \leqslant i$.
4: Swap $\pi[j]$ and $\pi[i]$.
5: **end for**

Algorithm 2 is due to Durstenfeld [9], it is usually called Fisher-Yates shuffle [18]. There are several variants of Algorithm 2 to generate a random permutation. However, indeed, the asymptotic time complexity of Algorithm 2 has already been optimal. This is the reason for this algorithm to be used for random permutation generation in this work.

3.2. MMC encryption

Next, we describe Procedure MMC-Encryption.

Algorithm 3. Procedure MMC-Encryption

Input: The original MMC problem Φ and the secret key K: $\{\alpha_1, \ldots, \alpha_m\}, \{\beta_1, \ldots, \beta_n\}, \{\gamma_1, \ldots, \gamma_s\}, \pi_1, \pi_2, \pi_3$.
Output: $\Phi_K = (\mathbf{X}', \mathbf{Y}')$.
1: The client generates matrices $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_2$, where $\mathbf{P}_1(i,j) = \alpha_i \delta_{\pi_1(i),j}, \mathbf{P}_2(i,j) = \beta_i \delta_{\pi_2(i),j}, \mathbf{P}_3(i,j) = \gamma_i \delta_{\pi_3(i),j}$.
2: The client computes $\mathbf{X}' = \mathbf{P}_1 \mathbf{X} \mathbf{P}_2^{-1}$ and $\mathbf{Y}' = \mathbf{P}_2 \mathbf{Y} \mathbf{P}_3^{-1}$. According to Theorems 1 and 2, the client can use (4) and (9) to efficiently (via time $O(n^2)$) compute $\mathbf{X}'$ and $\mathbf{Y}'$.
3: Later, the encrypted MMC problem $\Phi_K = (\mathbf{X}', \mathbf{Y}')$ will be outsourced to the cloud.

Lemma 1. *In Procedure MMC-Encryption, matrices* $\mathbf{P}_1, \mathbf{P}_2$, *and* $\mathbf{P}_3$ *are invertible. More precisely,*

$$\begin{cases} \mathbf{P}_1^{-1}(i,j) = (\alpha_j)^{-1}\delta_{\pi_1^{-1}(i),j}, \\ \mathbf{P}_2^{-1}(i,j) = (\beta_j)^{-1}\delta_{\pi_2^{-1}(i),j}, \\ \mathbf{P}_3^{-1}(i,j) = (\gamma_j)^{-1}\delta_{\pi_3^{-1}(i),j}. \end{cases} \tag{3}$$

Proof. Since $0 \notin \mathcal{K}_\alpha$, the determinant of $\mathbf{P}_1$ satisfies $\det(\mathbf{P}_1) \neq 0$. Hence, $\mathbf{P}_1$ is invertible. Likewise, $\mathbf{P}_2$ and $\mathbf{P}_3$ are invertible. Hereafter, the proof is straightforward and therefore is omitted. □

Theorem 1. *In Procedure MMC-Encryption, if* $\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1}$, *then it holds that*

$$\mathbf{X}'(i,j) = (\alpha_i/\beta_j)\mathbf{X}(\pi_1(i), \pi_2(j)). \tag{4}$$

Proof. Let

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix}. \tag{5}$$

Since $\mathbf{P}_1(i,j) = \alpha_i\delta_{\pi_1(i),j}$, this leads to

$$\mathbf{P}_1\mathbf{X} = \begin{bmatrix} \alpha_1 x_{\pi_1(1),1} & \cdots & \alpha_1 x_{\pi_1(1),n} \\ \vdots & \ddots & \vdots \\ \alpha_i x_{\pi_1(i),1} & \cdots & \alpha_i x_{\pi_1(i),n} \\ \vdots & \ddots & \vdots \\ \alpha_m x_{\pi_1(m),1} & \cdots & \alpha_m x_{\pi_1(m),n} \end{bmatrix}. \tag{6}$$

By Lemma 1, we have $\mathbf{P}_2^{-1}(i,j) = (\beta_j)^{-1}\delta_{\pi_2^{-1}(i),j}$. Then, one can obtain

$$\mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1} = \begin{bmatrix} \frac{\alpha_1}{\beta_1} x_{\pi_1(1),\pi_2(1)} & \cdots & \frac{\alpha_1}{\beta_j} x_{\pi_1(1),\pi_2(j)} & \cdots & \frac{\alpha_1}{\beta_n} x_{\pi_1(1),\pi_2(n)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\alpha_i}{\beta_1} x_{\pi_1(i),\pi_2(1)} & \cdots & \frac{\alpha_i}{\beta_j} x_{\pi_1(i),\pi_2(j)} & \cdots & \frac{\alpha_i}{\beta_n} x_{\pi_1(i),\pi_2(n)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{\alpha_m}{\beta_1} x_{\pi_1(m),\pi_2(1)} & \cdots & \frac{\alpha_m}{\beta_j} x_{\pi_1(m),\pi_2(j)} & \cdots & \frac{\alpha_m}{\beta_n} x_{\pi_1(m),\pi_2(n)} \end{bmatrix}. \tag{7}$$

This can be finally rewritten as

$$\mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1} = \mathbf{X}'(i,j) = (\alpha_i/\beta_j)\mathbf{X}(\pi_1(i), \pi_2(j)). \tag{8}$$

The proof is completed. □

Analogously, the following theorem is obtained.

Theorem 2. *In Procedure MMC-Encryption, if* $\mathbf{Y}' = \mathbf{P}_2'\mathbf{Y}\mathbf{P}_3^{-1}$, *then it follows that*

$$\mathbf{Y}'(i,j) = (\beta_i/\gamma_j)\mathbf{Y}(\pi_2(i), \pi_3(j)). \tag{9}$$

3.3. MMC in the cloud

See Procedure Φ_K-in-the-Cloud.

Algorithm 4. Procedure Φ_K-in-the-Cloud

Input: $\Phi_K = (\mathbf{X}', \mathbf{Y}')$.
Output: $\mathbf{Z}' = \mathbf{X}'\mathbf{Y}'$.
1: On input the encrypted MMC problem $\Phi_K = (\mathbf{X}', \mathbf{Y}')$, the cloud then invokes any matrix multiplication algorithm to compute $\mathbf{Z}' = \mathbf{X}'\mathbf{Y}'$.
2: The cloud then sends matrix $\mathbf{Z}'$ back to the client.

3.4. MMC decryption

See Procedure MMC-Decryption.

Algorithm 5. Procedure MMC-Decryption

Input: $\mathbf{Z}'$ and the secret key K.
Output: $\mathbf{Z}$.
1: On receiving the returned matrix $\mathbf{Z}'$ from the cloud, the client compute $\mathbf{Z} = \mathbf{P}_1^{-1}\mathbf{Z}'\mathbf{P}_3$. According to Theorem 3, the client can use (10) to efficiently (via time $O(n^2)$) compute $\mathbf{Z}$.

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 [10,28]. Technique details are elaborated in Procedure Result-Verification. We defer the detailed analysis of it in Section 4.

Algorithm 6. Procedure Result-Verification

Input: The decrypted but unchecked result $\mathbf{Z}$.
Output: Accepts $\mathbf{Z}$ as the correct result; or rejects it.
1: **for** $i = 1 : l$
2: The client generates an $s \times 1$ random 0/1 vector $\mathbf{r}$.
3: The client computes $\mathbf{P} = \mathbf{X} \times (\mathbf{Yr}) - \mathbf{Z} \times \mathbf{r}$.
4: **if** $\mathbf{P} \neq (0, \ldots, 0)^{\mathrm{T}}$
5: Output "verification fails"; aborts.
6: **end if**
7: **end for**
8: The client accepts $\mathbf{Z}$ as a correct result if it passes the above check, or else, it will be rejected.

Theorem 3. *In Procedure Result-Verification, if* $\mathbf{Z} = \mathbf{P}_1^{-1}\mathbf{Z}'\mathbf{P}_3$, *then it holds that*

$$\mathbf{Z}(i,j) = (\gamma_{\pi_3^{-1}(j)}/\alpha_{\pi_1^{-1}(i)})\mathbf{Z}'\left(\pi_1^{-1}(i), \pi_3^{-1}(j)\right). \tag{10}$$

Proof. The detailed proof is similar to Theorem 1. We now briefly describe the proof. By Lemma 1, we have $\mathbf{P}_1^{-1}(i,j) = (\alpha_j)^{-1}\delta_{\pi_1^{-1}(i)j}$. Together with $\mathbf{P}_3(i,j) = \gamma_i\delta_{\pi_3(i)j}$, then (10) can be deduced from $\mathbf{Z} = \mathbf{P}_1^{-1}\mathbf{Z}'\mathbf{P}_3$. □

3.6. The completed protocol

We now present the completed protocol that contains five sub-algorithms (KeyGen, MMCEnc, MMCSolve, MMCDec, ResultVerify) as follows:

- KeyGen(1^λ): On input a security parameter λ, the client invokes Procedure Secret-Key-Generation to get a secret key $K : \{\alpha_1, \ldots, \alpha_m\}, \{\beta_1, \ldots, \beta_n\}, \{\gamma_1, \ldots, \gamma_s\}, \pi_1, \pi_2, \pi_3$.
- MMCEnc($\Phi; K$): On input the original MMC problem Φ and the secret key K, the client invokes Procedure MMC-Encryption to encrypt Φ into an encrypted MMC Φ_K to protect input privacy.
- MMCSolve(Φ_K): On input the encrypted MMC problem Φ_K, the cloud invokes Procedure Φ_K-in-the-Cloud to get a result $\mathbf{Z}'$ to Φ_K. Then, the cloud returns $\mathbf{Z}'$ and an empty proof Γ to the client.
- MMCDec($\mathbf{Z}', K$): On input the returned result $\mathbf{Z}'$ and the secret key K, the client invokes Procedure MMC-Decryption to get the unchecked result $\mathbf{Z}$ to the original MMC problem Φ.
- ResultVerify($\mathbf{Z}, \Gamma$): On input the decrypted but unchecked result $\mathbf{Z}$ and the empty proof Γ, the client invokes Procedure Result-Verification to check its correctness. If they pass the check, then accepts it as the correct result; otherwise, just rejects it.

3.7. A numerical example

A numerical example is presented to illustrate the encryption and decryption processes. Suppose that the client intends to outsource a MMC problem $\Phi = (\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ and $\mathbf{Y} = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$. The client generates the secret key to be: $\{\alpha_1, \alpha_2\} = \{2, 3\}$, $\{\beta_1, \beta_2, \beta_3\} = \{4, 5, 6\}$, $\{\gamma_1, \gamma_2, \gamma_3, \gamma_4\} = \{7, 8, 9, 10\}, \pi_1 = [2, 1], \pi_2 = [2, 1, 3], \pi_3 = [1, 3, 4, 2]$. Then one has $\mathbf{P}_1 = \begin{bmatrix} 0 & 2 \\ 3 & 0 \end{bmatrix}, \mathbf{P}_1^{-1} = \begin{bmatrix} 0 & 1/3 \\ 1/2 & 0 \end{bmatrix}$, $\mathbf{P}_2 = \begin{bmatrix} 0 & 4 & 0 \\ 5 & 0 & 0 \\ 0 & 0 & 6 \end{bmatrix}$, $\mathbf{P}_2^{-1} = \begin{bmatrix} 0 & 1/5 & 0 \\ 1/4 & 0 & 0 \\ 0 & 0 & 1/6 \end{bmatrix}$, $\mathbf{P}_3 = \begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 9 \\ 0 & 10 & 0 & 0 \end{bmatrix}$, and $\mathbf{P}_3^{-1} = \begin{bmatrix} 1/7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/10 \\ 0 & 1/8 & 0 & 0 \\ 0 & 0 & 1/9 & 0 \end{bmatrix}$. The client then computes the encrypted MMC problem $\Phi_K = (\mathbf{X}', \mathbf{Y}')$, where $\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1} = \begin{bmatrix} 1/2 & 2/5 & 1/3 \\ 3/4 & 3/5 & 1/2 \end{bmatrix}$ and $\mathbf{Y}' = \mathbf{P}_2\mathbf{Y}\mathbf{P}_3^{-1} = \begin{bmatrix} 8/7 & 1 & 8/9 & 4/5 \\ 10/7 & 5/4 & 10/9 & 1 \\ 12/7 & 3/2 & 4/3 & 6/5 \end{bmatrix}$. On input the encrypted MMC problem $\Phi_K = (\mathbf{X}', \mathbf{Y}')$, the cloud computes $\mathbf{Z}' = \mathbf{X}'\mathbf{Y}' = \begin{bmatrix} 12/7 & 3/2 & 4/3 & 6/5 \\ 18/7 & 9/4 & 2 & 9/5 \end{bmatrix}$ and then sends it back to the client. On input the returned result $\mathbf{Z}'$, the client recovers the result to the original MMC problem by computing $\mathbf{Z} = \mathbf{P}_1^{-1}\mathbf{Z}'\mathbf{P}_3 = \begin{bmatrix} 6 & 6 & 6 & 6 \\ 6 & 6 & 6 & 6 \end{bmatrix}$. It can be easily verified that $\mathbf{Z} = \mathbf{X}\mathbf{Y}$.

The above example can help the readers to gain an insightful understanding of the proposed protocol.

4. Correctness, security, and verifiability analysis

4.1. Correctness guarantee

Theorem 4. *The proposed protocol is correct.*

Proof. It suffices to show that if both the client and the cloud follow the protocol honestly, then the result $\mathbf{Z}'$ returned by a honest cloud server will always be decrypted successfully and the corresponding result $\mathbf{Z}$ is always correct. Observe that $\mathbf{X}'$ and $\mathbf{Y}'$ are given by $\mathbf{X}' = \mathbf{P}_1\mathbf{X}\mathbf{P}_2^{-1}, \mathbf{Y}' = \mathbf{P}_2\mathbf{Y}\mathbf{P}_3^{-1}$. Note that an honest cloud server computes $\mathbf{Z}' = \mathbf{X}'\mathbf{Y}' = \mathbf{P}_1\mathbf{X}\mathbf{Y}\mathbf{P}_3^{-1}$, then by $\mathbf{Z} = \mathbf{P}_1^{-1}\mathbf{Z}'\mathbf{P}_3$, we have $\mathbf{Z} = \mathbf{X}\mathbf{Y}$. This implies the proposed protocol is correct. □

4.2. Security guarantee

4.2.1. Input privacy

The proposed protocol can protect input privacy if given the encrypted Φ_K, the cloud cannot recover the client's input data in Φ. We first consider the case given the encrypted matrix $\mathbf{X}'$, the cloud attempts to recover the original matrix $\mathbf{X}$. The original matrix $\mathbf{X}$ is encrypted by the following two phases:

1: The position of each entry in the original matrix $\mathbf{X}$ is rearranged under two random permutations, i.e., $\mathbf{T}(i,j) = \mathbf{X}(\pi_1(i), \pi_2(j))$.
2: Each entry in matrix $\mathbf{T}$ is further masked by multiplying a factor, i.e., $\mathbf{X}'(i,j) = (\alpha_i/\beta_j)\mathbf{T}(i,j)$.

In Phase 1, the key space consists of all random permutations of π_1, π_2, meaning that there are $m! \cdot n!$ cases of permutations. Each case occurs exactly with probability $\frac{1}{m! \cdot n!}$. This implies that even if the cloud has the correct matrix $\mathbf{T}$, the expected time of brute-force attack on the key space to recover the original matrix $\mathbf{X}$ is $\frac{m! \cdot n!}{2}$, which is definitely a non-polynomially-bounded quantity. In Phase 2, each entry in matrix $\mathbf{T}$ is further masked, the expected time of brute-force attack on the key space to guess $\{\alpha_1, \ldots, \alpha_m\}$ and $\{\beta_1, \ldots, \beta_n\}$ is $\frac{|\mathcal{K}_\alpha|^m \cdot |\mathcal{K}_\beta|^n}{2}$. A choice of large key spaces $\mathcal{K}_\alpha$ and $\mathcal{K}_\beta$ will thwart this attack. Further consider that a new different secret key is generated in each run of the protocol, so the cloud cannot recover $\mathbf{X}$ from $\mathbf{X}'$ by trivial means. Likewise, the cloud cannot recover $\mathbf{Y}$ from $\mathbf{Y}'$. Accordingly, the proposed protocol is believed to reach an applicable secure 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 result $\mathbf{Z}'$, the cloud cannot recover the correct result $\mathbf{Z}$ to the original MMC problem Φ. Based on (10), we have that the protection of output privacy can be analyzed in the same way with that of input privacy. It is omitted accordingly.

4.3. Verifiability guarantee

Theorem 5. *The proposed protocol satisfies robust cheating resistance.*

Proof. The correctness of the decrypted result $\mathbf{Z}$ is checked from Step 1 to Step 7 in Procedure Result-Verification. The random check process from Step 2 to Step 6 is repeated l times. We now define the following two parameters to facilitate our proof. Let $Prob_1$ be the probability of non-detection of a false returned result in one round of a random check process. Let $Prob_f$ denote the probability of *check failure*, i.e., the probability of non-detection of a false returned result in whole l times random check processes.

The proof consists of two steps. First, we show that the result from a faithful cloud server must be verified successfully by the client. From Theorem 4, if the cloud is faithful, we have $\mathbf{Z} = \mathbf{XY}$. So

$$\mathbf{P} = \mathbf{X} \times (\mathbf{Yr}) - \mathbf{Z} \times \mathbf{r} = (0, \ldots, 0)^{\mathrm{T}}, \tag{11}$$

regardless of what vector $\mathbf{r}$ is. Therefore, the verification failure step (Step 5 in Procedure Result-Verification) will never be executed. This means that a correct result $\mathbf{Z}$ must be verified successfully by the client.

Next, we show that no false result 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. Let

$$\mathbf{D} = \mathbf{X} \times \mathbf{Y} - \mathbf{Z}, \mathbf{P} = \mathbf{D} \times \mathbf{r} = (p_1, \ldots, p_m)^{\mathrm{T}}, \tag{12}$$

if the cheating cloud return a false $\mathbf{Z}'$, then this leads to $\mathbf{Z} \neq \mathbf{XY}$. Accordingly, we have $\mathbf{D} \neq \mathbf{0}$, so at least one element of $\mathbf{D}$ is non-zero. Suppose that the element $d_{ij} \neq 0$. By the definition of matrix–vector multiplication,

$$p_i = \sum_{k=1}^{s} d_{ik} r_k = d_{i1} r_1 + \cdots + d_{ij} r_j + \cdots + d_{is} r_s = d_{ij} r_j + y, \tag{13}$$

where $y = \sum_{k=1}^{s} d_{ik} r_k - d_{ij} r_j$. Then, Total Probability Theorem yields

$$\Pr[p_i = 0] = \Pr[p_i = 0 | y = 0] \Pr[y = 0] + \Pr[p_i = 0 | y \neq 0] \Pr[y \neq 0]. \tag{14}$$

Note from (13) that

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

Substituting (15) into (14) results in

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

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

$$\Pr[p_i = 0] \leqslant 1/2. \tag{17}$$

Based on (17) $Prob_1$ satisfies

$$Prob_1 = \Pr[\mathbf{P} = (0, \ldots, 0)^{\mathrm{T}}] \leqslant \Pr[p_i = 0] \leqslant \frac{1}{2}. \tag{18}$$

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

$$Prob_f \leqslant Prob_1^l \leqslant \frac{1}{2^l}, \tag{19}$$

which demonstrates that $Prob_f$ is a negligible quantity in terms of l. This completes the proof. □

It can be deduced from the proof of Theorem 5 that the proposed protocol can handle result verification with check failure (non-detection of false result) probability at most 2^{-l}. The size of l is a tradeoff between the probability of cheating failure and efficiency. While a conservative choice of high cheating resistance should probably require l to be around 80 bits, for a fast check a reasonable choice of 20 bits is also acceptable.

4.4. Further discussions on result verification

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

Definition 1. (Monte Carlo Verification Algorithm [28]) The classification and definition of Monte Carlo verification algorithm is summarized in Table 1. The detailed verbal description of case 1 is as follows: for a randomized verification algorithm Vrfy and any decrypted but unchecked result *Res*, if

Table 1
Classification and definition of Monte Carlo verification algorithm.

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

$$\begin{aligned} &\Pr[\text{Vrfy accepts } Res \mid Res \text{ is correct}] = 1, \\ &\Pr[\text{Vrfy accepts } Res \mid Res \text{ is false}] \leqslant \delta, \end{aligned} \tag{20}$$

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

Based on Definition 1 and the proof of Theorem 5, we immediately have the following theorem.

Theorem 6. *One round of random check process, i.e., from Step 2 to Step 6 in Procedure Result-Verification, 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 (and the success probability amplified) by running the algorithm multiple times. Indeed, this mechanism has been exploited in our Procedure Result-Verification. According to the above analysis, case 2 and case 3 of Monte Carlo verification algorithms (see Table 1) 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 handling result verification, which is usually 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, MMCEnc, MMCDec, and ResultVerify. The time cost for KeyGen is $O(m+n+s)$. In MMCEnc, applying (4) and (9) to efficiently compute $\mathbf{X}'$ and $\mathbf{Y}'$, it only takes time $O(mn+ns)$. Likewise, the time consumed by MMCDec is $O(ms)$. As to ResultVerify, it only takes time $O(ms)$.

Cloud side overhead. For the cloud, its only computation overhead is generated by running MMCSolve. Suppose that the schoolbook matrix multiplication is used in the cloud side, the time consumed by MMCSolve is $O(mns)$.

Table 2 summarizes the theoretical performance of the proposed protocol. The overall time cost is $O(mn+ns+ms)$ for the client and $O(mns)$ for the cloud. From the perspective of efficiency, the proposed protocol is feasible due to there exists a gap between $O(mn+ns+ms)$ and $O(mns)$. Therefore, as long as (m,n,s) become sufficiently large, the proposed protocol is able to allow the client to outsource MMC to the cloud and gain substantial computational savings. This claim will be further validated by our experiments in the next subsection.

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 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 will be case-specific, depending on the asymmetric computing speed owned by the two different workstations. Consequently, one-workstation-based

Table 2
Theoretical performance of the proposed protocol.

Client side					Cloud side	
KeyGen	MMCEnc	MMCDec	ResultVerify	Sending cost	MMCSolve	Sending cost
$O(m+n+s)$	$O(mn+ns)$	$O(ms)$	$O(ms)$	$\Phi_K = (\mathbf{X}', \mathbf{Y}')$	$O(mns)$	$\mathbf{Z}'$

Table 3
Notations.

Notations	Means
t_{original}	The time for the client to compute the original MMC locally
t_{cloud}	The time for the cloud to compute the outsourced MMC
t_{client1}	The time for the client to generate the secret key and encrypt the original MMC
t_{client2}	The time for the client to decrypt and verify the returned result
t_{client}	$t_{\text{client}} = t_{\text{client1}} + t_{\text{client2}}$

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 3, the performance gain of the client can be shown by $\frac{t_{\text{original}}}{t_{\text{client}}}$; we refer to this as *client speedup*. This value theoretically should be a considerable positive number greater than 1, which means there is a considerable performance gain. We also consider another metric, i.e., the *cloud efficiency*, using $\frac{t_{\text{original}}}{t_{\text{cloud}}}$. Ideally, the MMC encryption should not increase the time to solve the original MMC. It is desirable that the cloud efficiency is close to 1.

The implementation is done using Matlab2012a on a workstation equipped with Intel (R) Core (TM) 3.20 GHz CPU and 4 GB RAM. The schoolbook matrix multiplication algorithm is used by the cloud. We set $m : n : s = 4 : 5 : 6$, $\mathcal{K}_\alpha = \{1, \ldots, m\}, \mathcal{K}_\beta = \{1, \ldots, n\}$, and $\mathcal{K}_\gamma = \{1, \ldots, s\}$. All of matrix instances in experiments are generated with each entry randomly located in $(0, 1)$. Three groups of experiments are conducted with $l = 20, l = 50$, and $l = 80$, which correspond to efficiency priority, tradeoff, and cheating resistance priority cases.

The main performance is shown in Table 4. It can be observed that client speedup is monotonically increasing with matrix dimensions. Outsourcing MMC is able to gain more than 14 times client speedup if (m, n, s) are sufficiently large. It is shown from Table 4 that $t_{\text{client2}} > t_{\text{client1}}$, which indicates that handling result decryption and verification is more costly. Besides, the cloud efficiency stays close to 1, which is very satisfactory. The comparison of client speedup as a function of n is depicted in Fig. 2. It is shown that client speedup in the case of efficiency priority ($l = 20$) is much larger than that in the case of cheating resistance priority ($l = 80$), which is expected. Note that, in our experiments, matrix dimensions are no more than 3600. A matrix with dimensions no more than 3600 is not an unreasonably large matrix. Many real world applications, e.g., MMC from scientific computing, could easily lead to large matrices with considerably more than 3600 dimensions.

6. Related work and comparisons

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

Table 4
Performance of the proposed protocol (time is in "seconds").

Benchmark		Original MMC	Encrypted MMC_K				Client speedup	Cloud efficiency
No.	(m, n, s)	t_{original}	t_{cloud}	t_{client1}	t_{client2}	t_{client}	$t_{\text{original}}/t_{\text{client}}$	$t_{\text{original}}/t_{\text{cloud}}$
$l = 20$ *with check failure* $Prob_f \leqslant \frac{1}{2^{20}}$: **Low cheating resistance and high client speedup**								
1	(200, 250, 300)	0.3364	0.3389	0.0393	0.1164	0.1558	2.1597×	0.9927
2	(400, 500, 600)	2.6455	2.6606	0.1392	0.5419	0.6811	3.8840×	0.9943
3	(800, 1000, 1200)	26.1812	26.5996	0.9490	2.2146	3.1636	8.2757×	0.9843
4	(1600, 2000, 2400)	236.8153	239.2615	8.2997	11.4307	19.7304	12.0026×	0.9898
5	(2400, 3000, 3600)	826.1342	833.2995	28.8096	29.9363	58.7460	14.0628×	0.9914
$l = 50$ *with check failure* $Prob_f \leqslant \frac{1}{2^{50}}$: **Tradeoff between cheating resistance and client speedup**								
1	(200, 250, 300)	0.3340	0.3353	0.0512	0.2674	0.3186	1.0483×	0.9959
2	(400, 500, 600)	2.6595	2.6938	0.1507	1.2681	1.4187	1.8746×	0.9873
3	(800, 1000, 1200)	26.2342	27.3243	0.9739	5.0258	5.9997	4.3726×	0.9601
4	(1600, 2000, 2400)	239.8311	242.0428	8.3302	22.6239	30.9541	7.7480×	0.9909
5	(2400, 3000, 3600)	825.9332	831.2559	28.8230	56.6145	85.4375	9.6671×	0.9936
$l = 80$ *with check failure* $Prob_f \leqslant \frac{1}{2^{80}}$: **High cheating resistance and low client speedup**								
1	(200, 250, 300)	0.3302	0.3395	0.0442	0.4131	0.4573	0.7221×	0.9726
2	(400, 500, 600)	2.5496	2.5970	0.1433	1.9590	2.1023	1.2128×	0.9818
3	(800, 1000, 1200)	26.4048	27.1520	0.9285	8.0222	8.9508	2.9500×	0.9725
4	(1600, 2000, 2400)	236.5965	237.5374	8.3211	34.0214	42.3424	5.5877×	0.9960
5	(2400, 3000, 3600)	827.0486	837.0930	29.7640	82.8537	112.6177	7.3439×	0.9880

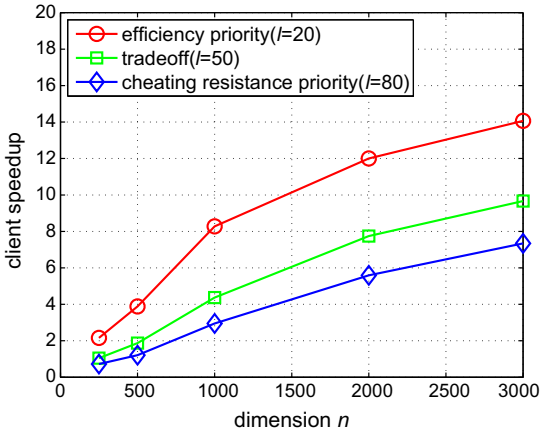

Fig. 2. Comparison of client speedup ($m : n : s = 4 : 5 : 6$).

secure outsourcing is persistently increasing, especially after Gentry's first FHE scheme [12] by using an ideal lattice. They often focus on designing a generic protocol that covers all problems, e.g., [11,8]. 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 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. [3] proposes a number of protocols for secure outsourcing scientific computations, such as solving a linear system of equations, sorting, etc. They employ a lot of problem transformation techniques to construct the protocols, but the common drawbacks of their protocols are twofold: 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 [5,2]. The former is built upon the assumptions of two non-colluding servers, making it vulnerable to colluding attacks. While the later achieves provable security using Shamir's secret sharing [30] technique. But this theoretically elegant protocol only works over finite field $\mathbb{Z}_p$ and it suffers from large amount of communication overhead. A comprehensive comparative study on these three protocols is shown in Table 5. 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, handling result verification becomes a must. Following this trend, several protocols that can handle result verification are proposed, among which there are the secure outsourcing of linear programming [33], the secure outsourcing of linear equations [34], and the secure outsourcing of matrix inversion [19], etc. Recently, the works in [33,34] are further improved in [7] by employing some special linear transformations and a pseudorandom number generator. The work in [19] sheds some insight on constructing secure MMC outsourcing protocol, but matrix inversion outsourcing and matrix multiplication outsourcing are different in several aspects. Consider, for example, there is only one input in matrix inversion computation, whereas there are two inputs in matrix multiplication computation. Besides, the input of matrix inversion must be a square matrix, and the inputs of matrix multiplication may be two non-square matrices. Frankly, our system model and framework are inherited from these works.

Table 5
Comparisons with other protocols.

Properties	Our protocol	Protocol [5]	Protocol [2]
The underlying technique	Permutation	Homomorphic encryption	Secret sharing
Algebraic structure of matrix element	Infinite field $\mathbb{R}$	Infinite field $\mathbb{R}$	Finite field $\mathbb{Z}_p$
Dimensions of two matrices	Multiplicable matrices	Square matrices	Square matrices
Number of remote servers	One	Two	One or two
Require two servers non-colluding	No	Yes	No
Handle result verifiability	Yes	Yes	Yes
Provide experimental evaluation	Yes	No	No
Achieve provable security	No	Yes	Yes

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 [37]. SMC does not consider the asymmetry between the resources possessed by cloud and client, and therefore, it cannot be applied to secure outsourcing directly. The second one is about delegating computation and cheating detection, e.g., [14]. 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 [25,15,17]. 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

In retrospect, we have designed a protocol for outsourcing of MMC to a malicious cloud. We have shown that the proposed protocol simultaneously fulfills goals of correctness, security (input/output privacy), robust cheating resistance, and high-efficiency. With MMC already well rooted in scientific and engineering fields, the proposed protocol can be deployed individually or serve as a primitive building block, based on which some higher level secure outsourcing protocols are constructed. We also introduced a Monte Carlo verification algorithm. Its superiority in designing inexpensive result verification algorithm for secure outsourcing is well demonstrated. Directions to launch further research include: (1) establishing formal security framework for MMC outsourcing problem; (2) exploring an encryption scheme that can further reduce the computational overhead at the client side; (3) adding result verification for some early protocols, which do not handle result verification, as a counter offensive to malicious cloud.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (61170249), the Natural Science Foundation Project of CQCSTC (2009BA2024), in part by the program for Changjiang scholars, in part by Specialized Research Fund for priority areas for the Doctoral Program of Higher Education and in part by the Research Fund of Preferential Development Domain for the Doctoral Program of Ministry of Education of China under Grant 201101911130005. This publication was made possible by NPRP Grant 4-1162-1-181 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[2] M.J. Atallah, K.B. Frikken, Securely outsourcing linear algebra computations, in: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ACM, 2010, pp. 48–59.
[3] M.J. Atallah, K.N. Pantazopoulos, J.R. Rice, E.E. Spafford, Secure outsourcing of scientific computations, Adv. Comput. 54 (2002) 215–272.
[4] Michael Basin, Pablo Rodriguez-Ramirez, Sliding mode filter design for nonlinear polynomial systems with unmeasured states, Inform. Sci. 204 (2012) 82–91.
[5] David Benjamin, Mikhail J Atallah, Private and cheating-free outsourcing of algebraic computations, in: Sixth Annual Conference on Privacy, Security and Trust, 2008 (PST'08), IEEE, 2008, pp. 240–245.
[6] Glenn Brunette, Rich Mogull, Security guidance for critical areas of focus in cloud computing v2. 1. Cloud Security Alliance, 2009, pp. 1–76.
[7] Fei Chen, Tao Xiang, Yuanyuan Yang, Privacy-preserving and verifiable protocols for scientific computation outsourcing to the cloud, J. Parallel Distrib. Comput. 74 (3) (2014) 2141–2151.
[8] K.M. Chung, Y. Kalai, S. Vadhan, Improved delegation of computation using fully homomorphic encryption, Advances in Cryptology–CRYPTO 2010, 2010, pp. 483–501.
[9] Richard Durstenfeld, Algorithm 235: random permutation, Commun. ACM 7 (7) (1964) 420–421.
[10] Rusins Freivalds, Probabilistic machines can use less running time, Inform. Process. 77 (1977) 839–842.
[11] R. Gennaro, C. Gentry, B. Parno, Non-interactive verifiable computing: outsourcing computation to untrusted workers, Advances in Cryptology–CRYPTO 2010, 2010, pp. 465–482.
[12] C. Gentry, A fully homomorphic encryption scheme, PhD thesis, Stanford University, 2009.
[13] Sarah F.F. Gibson, Brian Mirtich, A survey of deformable modeling in computer graphics, Technical Report TR-97-19, 1997.
[14] S. Goldwasser, Y.T. Kalai, G.N. Rothblum, Delegating computation: interactive proofs for muggles, in: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, ACM, 2008, pp. 113–122.
[15] S. Hohenberger, A. Lysyanskaya, How to securely outsource cryptographic computations, Theory Cryptogr. 3378 (2005) 264–282.
[16] Tinggang Jia, Yugang Niu, Yuanyuan Zou, Sliding mode control for stochastic systems subject to packet losses, Inform. Sci. 217 (2012) 117–126.
[17] S. Kawamura, A. Shimbo, Fast server-aided secret computation protocols for modular exponentiation, IEEE J. Select. Areas Commun. 11 (5) (1993) 778–784.
[18] Donald Ervin Knuth, The Art of Computer Programming, addison-Wesley, 2006.
[19] Xinyu Lei, Xiaofeng Liao, Tingwen Huang, Huaqing Li, Chunqiang Hu, Outsourcing large matrix inversion computation to a public cloud, IEEE Trans. Cloud Comput. 1 (1) (2013) 78–87.
[20] Yehuda Lindell, Benny Pinkas, Secure multiparty computation for privacy-preserving data mining, J. Privacy Confident. 1 (1) (2009) 59–98.
[21] Hongbo Liu, Ajith Abraham, Václav Snášel, Seán McLoone, Swarm scheduling approaches for work-flow applications with security constraints in distributed data-intensive computing environments, Inform. Sci. 192 (2012) 228–243.
[22] Qin Liu, Guojun Wang, Jie Wu, Time-based proxy re-encryption scheme for secure data sharing in a cloud environment, Inform. Sci. 258 (2014) 355–370.
[23] Jaime Lloret, Miguel Garcia, Jesus Tomas, Joel JPC Rodrigues, Architecture and protocol for intercloud communication, Inform. Sci. 258 (2014) 434–451.

[24] Gregorio Martinez, Sherali Zeadally, Han-Chieh Chao, Editorial: cloud computing service and architecture models, Inform. Sci.: An Int. J. 258 (2014) 353–354.
[25] T. Matsumoto, K. Kato, H. Imai, Speeding up secret computations with insecure auxiliary devices, in: Advances in Cryptology–CRYPTO 88, Springer, 1990, pp. 497–506.
[26] Carl Meyer, Matrix Analysis and Applied Linear Algebra Book and Solutions Manual, vol. 2, Society for Industrial and Applied Mathematics, 2000.
[27] Mehdi Mohammadi, Bijan Raahemi, Ahmad Akbari, Babak Nassersharif, Hossein Moeinzadeh, Improving linear discriminant analysis with artificial immune system-based evolutionary algorithms, Inform. Sci. 189 (2012) 219–232.
[28] Rajeev Motwani, Prabhakar Raghavan, Randomized Algorithms, Cambridge university press, 1995.
[29] George AF. Seber, Alan J. Lee, Linear Regression Analysis, vol. 936, Wiley, 2012.
[30] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.
[31] Ran Tao, Xiang-Yi Meng, Yue Wang, Image encryption with multiorders of fractional fourier transforms, IEEE Trans. Inform. Forensics Sec. 5 (4) (2010) 734–738.
[32] Juan Vera-del Campo, Josep Pegueroles, Juan Hernández-Serrano, Miguel Soriano, Doccloud: a document recommender system on cloud computing with plausible deniability, Inform. Sci. 258 (2014) 387–402.
[33] Cong Wang, Kui Ren, Jia Wang, Secure and practical outsourcing of linear programming in cloud computing, in: INFOCOM, 2011 Proceedings, IEEE, 2011, pp. 820–828.
[34] Cong Wang, Qian Wang, Kui Ren, Jia Wang, Harnessing the cloud for securely outsourcing large-scale systems of linear equations, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1172–1181.
[35] Lifei Wei, Haojin Zhu, Zhenfu Cao, Xiaolei Dong, Weiwei Jia, Yunlu Chen, Athanasios V. Vasilakos, Security and privacy for storage and computation in cloud computing, Inform. Sci. 258 (2014) 371–386.
[36] Jie Xu, Jian Yang, Zhihui Lai, K-local hyperplane distance nearest neighbor classifier oriented local discriminant analysis, Inform. Sci. 232 (2013) 11–26.
[37] A.C. Yao, Protocols for secure computations, in: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, 1982, pp. 160–164.
[38] Zhi-liang Zhu, Wei Zhang, Kwok-wo Wong, Hai Yu, A chaos-based symmetric image encryption scheme using a bit-level permutation, Inform. Sci. 181 (6) (2011) 1171–1186.