Cluster Computing (2020) 23:2835–2845
https://doi.org/10.1007/s10586-020-03049-7

Secure and efficient publicly verifiable outsourcing of matrix multiplication in online mode

Fatemeh Erfan[1] · Hamid Mala[1]

Received: 30 April 2019 / Revised: 3 November 2019 / Accepted: 6 January 2020 / Published online: 23 January 2020

Abstract
With the emergence of cloud computing paradigm in many scientific applications, outsourcing of computation has attracted a great amount of attention from the research community. Outsourcing of heavy computations such as multiplication of two large matrices has raised some security concerns. Data and the result of computation should be protected not only from attackers, but also from the cloud servers. Moreover, data owner should be able to verify the correctness of computation with complexity less than the original computation. The previous schemes either have expensive offline phase or do not support public verifiability. In this paper, first we find a security vulnerability in the Zhang-Lei's scheme for outsourcing of matrix multiplication where the cloud server can forge the result and pass the verification phase. Then, we present a secure and efficient publicly verifiable outsourcing of matrix multiplication scheme which achieves privacy protection of outsourced data and result, unforgeability of result, public verifiability and high efficiency. Our analyses show that compared with the related work, the proposed scheme is superior in terms of functionality, computation, communication and storage overhead, especially in verification computation overhead.

Keywords Cloud computing · Matrix multiplication · Public verifiability · Secure outsourcing · Unforgeability · Privacy · Forgery

1 Introduction

With the explosive growth of data volume and computing scales in recent years, now many users and corporations are unable to afford their heavy storage and computing equipment with the same rate. Fortunately, cloud technology with its extensive storage and processing capacity offers an attractive solution.

Based on a pay-per-use model, a client with limited computational power can easily outsource large-scale computational tasks to a cloud server [31]. However, when the data to be outsourced is valuable and private, this solution is not so straightforward. The untrusted cloud may misuse or sell the data. Not only must the outsourced data be kept secret from the cloud but the result of computation over this private data also must be blind to the cloud. Therefore, the data owner (or client) and the cloud server need a security protocol that (1) protects the data not only against outsider eavesdroppers but also against the cloud, (2) allows the cloud to perform computation over this blinded data without finding the final result of its computation, (3) allows the data owner to efficiently discover the final result from the blinded result the cloud has computed, and (4) makes the data owner sure about the correctness of the final result. The latter requirement is known as verifiability. Verification of the result may be performed by the data owner itself or by any public verifier. The public verifier must not learn neither the original data nor the final result. In this paper, we focus on the secure outsourcing of multiplication of two large matrices of private entries. Our proposed protocol provides public verifiability.

✉ Hamid Mala
h.mala@eng.ui.ac.ir

Fatemeh Erfan
fatemeh.erfan@eng.ui.ac.ir

[1] Department of Information Technology Engineering, Faculty of Computer Engineering, University of Isfahan, Hezar Jerib St., Isfahan 81746-73441, Iran

1.1 Motivation

Matrix multiplication is a basic computational operation in many scientific and engineering fields and has numerous applications including computer graphics, image compression, image transformation and so on [15, 17]. A client with limited computational resources might be unable to tolerate huge overhead of multiplying large matrices. Therefore, it prefers to outsource matrix multiplication and verification of the results. On the other hand, the client wants to protect privacy of its sensitive data. In many cases, the client is unwilling to share the data with others, including the cloud server [31]. Also, we require that the total running time for the client, including blinding (or encrypting) the matrices, unblinding (or decrypting) the blinded result and verifying the correctness of the result, must be less than the time of the original computation. If not, the client would perform the computation on its own rather than outsource the computation [22].

The existing work mainly suffer from the massive overheads [3, 8, 10, 24]. In many previous schemes the client has to declare the matrices that he may require their multiplication in the future. In these schemes, the client declares two groups M_A and M_B of matrices, each identified with a unique ID, to the cloud server in the offline phase. Later, in the online phase, he is only allowed to request for the multiplication of a matrix from M_A by a matrix from M_B. Only a few schemes such as [19] allow the client to request for the multiplication of two fresh (not previously announced to the cloud server) matrices. As an instance of this, in Lei's scheme [19] the client can directly outsource its two matrices to the cloud server and request for the multiplication of these two fresh matrices. However, this scheme does not consider public verifiability. Moreover, some of them cannot protect the privacy of outsourced data and results [8, 10]. Therefore, we are motivated to design a secure and efficient protocol for outsourcing of multiplication of two freshly generated large matrices with public verifiability.

1.2 Challenges

Although it is quite promising, outsourcing computational problems to the commercial public clouds inevitably brings in new security concerns and challenges [19]. The first challenge is privacy of outsourced data and result. Generally speaking, the issue of security and privacy has become a major concern when the sensitive data is not processed in a fully trusted cloud environment [31]. Recently, a number of publications have been proposed to design specific secure schemes for outsourcing computation operations. In most of them data owner, i.e. a person who has sensitive data, tolerates massive overheads. But the privacy of sensitive data is not protected completely. In our scheme we consider that a client can outsource multiplication of two large matrices to a cloud server and even the verification of the result to a third party as a verifier.

The second challenge is verifying the result returned by the server. A cloud server might not always provide the correct result of a given computational task [19]. It might be lazy in order to save its resources and return random result to the data owner. In some cases, servers are not willing to report their failures and may return accidental results in order to keep their clients. Hence, the cloud servers need to prove their honesty by sending an unforgeable proof.

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. Conversely, the cloud may be unable to complete the task in a reasonable amount of time, or the cost of the cloud may become prohibitive [19]. In addition, most of the clients have no powerful resources to verify the result. Thus, a public verifiable outsourcing protocol with low computation overhead is needed.

1.3 Contribution

The contributions of this paper are summarized as below:

1. We present a forgery attack by the malicious cloud server against the Zhang-Lei's scheme for outsourcing matrix multiplication.
2. We propose a cryptographic protocol for secure outsourcing of multiplication of two arbitrary large matrices.
3. In our proposed scheme, the problem of high computation overhead in both client and verifier side is well addressed.
4. We show that the proposed protocol is efficient in terms of computation, communication and storage overhead.

1.4 Organization

The rest of the paper is organized as follows. Section 2 reviews related work on outsourcing of matrix multiplication to the cloud server. Section 3 describes the preliminaries including system model, threat model and design goals. In Sect. 4, we briefly review the Zhang-Lei's

scheme and present a security attack against this scheme. Section 5 presents our proposed scheme. Section 6 discusses correctness and security analysis of our scheme. In this section, we evaluate performance of our scheme and compare it with related work. Finally, Sect. 7 concludes the paper.

2 Related work

As previously stated, the problem of outsourcing computation has attracted extreme attention in academia. Recently, various papers has discussed secure outsourcing of heavy operations such as modular exponentiation [13, 22, 23, 28], bilinear pairing [5], Fourier transformation [26], matrix inversion [4, 14, 18], determinant computation [20] and matrix multiplication [16, 33]. In this paper, we focus on secure outsourcing of matrix multiplication. The schemes proposed for matrix multiplication in the literature can be classified into two categories as reviewed below.

Multiplication of two arbitrary matrices In some schemes [9, 15, 24, 30] two matrices involved in the protocol are variable. In fact, the client has two arbitrary matrices A and B and wants to outsource the computation of $A \times B$. These schemes are more flexible inasmuch as both input matrices involved in the protocol are of arbitrary, yet compatible, dimensions.

For the first time, in 2002, Atallah et al. investigated the problem of outsourced scientific computations like matrix multiplications and introduced an outsourcing computation framework. This schemes suffers from information leakage [1]. After that, in 2008, Benjamin et al. presented a private and cheating-free protocol for outsourcing algebraic computations where the data owner outsources its computations to two remote servers. One disadvantage of this technique is that the two servers have to be non-colluding. Moreover, the above protocol requires heavy computations that degrades the efficiency of the outsourcing process [3]. In 2010, Gennaro et al. proposed a scheme by combining fully homomorphic encryption (FHE) [12] and Yao's Garbled Circuits [27]. This scheme provides the formalized definition of non-interactive verifiable outsourcing computation using the pseudo-random functions (PRF) [11]. In addition, Atallah proposed protocols for secure outsourcing of matrix multiplication based on Shamir's secret sharing scheme. In this scheme the client outsources its data to at least two servers [2]. In 2011, Mohassel proposed a secure protocol for outsourcing linear algebra computation with verifiable results based on homomorphic encryption scheme that has high costs and overhead [24]. Zhang et al. presented a protocol for secure outsourcing of matrix multiplications. This approach has significant computation overhead in verification phase [32]. In 2017, a secure and efficient protocol is proposed for outsourcing large matrix multiplication in online mode. This scheme protects privacy of outsourced data and result. But it does not support public verification [9]. Recently, Zhang et al. designed a protocol for outsourcing matrix multiplication. This scheme has a massive offline phase [33]. Kong et al. presented a protocol of matrix multiplication based on similarity transformation. This scheme does not support public verification [17].

multiplication of an arbitrary matrix by a fixed matrix In some protocols [8, 10, 21, 29, 33], one of the two matrices must be constant. As a matter of fact, the client sends a constant matrix to the cloud server in offline phase, before starting an online phase. These schemes are not suitable for many real applications due to the massive communication overhead and runtime. In 2012, Fiore et al. presented a publicly verifiable protocol for outsourcing matrix multiplication based on bilinear pairing and algebraic PRFs. Based on [15], this approach not only fails to achieve high efficiency, but also fails to achieve data privacy [10]. In 2015, Jia et al. proposed a protocol for outsourcing matrix multiplication under amortized model. This scheme does not protect privacy of multiplication result and has a heavy preprocessing phase. Moreover, it does not provide public verification [15]. In 2016, Elkhiyaoui et al. proposed a protocol for outsourcing matrix multiplication. In this scheme, one of the matrices must be constant and the client tolerates massive computation [8].

3 Preliminaries

As mentioned above, our scheme is designed for publicly verifiable outsourcing of multiplication of two matrices with arbitrary sizes which is just executed in online mode. In Sect. 3.1, we present notation used in this paper and also the required mathematical background. In the next subsections, the system model, threat model and design goals of our scheme are clarified.

3.1 Notation and mathematical background

We denote notations used in this paper in Table 1. Based on [21, 32] we define the notion of bilinear pairing utilized in Zhang-Lei's scheme [30] which is explained in Sect. 4.

Definition 1 (*Bilinear Pairing*) Let G_1, G_2 and G_T be three multiplicative cyclic groups of the same prime order q, and g_1 and g_2 be a generators of group G_1 and G_2, respectively. The map $e : G_1 \times G_2 \rightarrow G_T$ is called a bilinear pairing if it satisfies the following properties.

– *Bilinearity*: for any $a, b \in Z_q$, any $g \in G_1$ and any $h \in G_2$ the equation $e(g^a, h^b) = e(g, h)^{ab}$ holds.

Table 1 Notations

Symbols	Meanings
A, B	The matrices before multiplication
$A_{i,j}$	The (i, j)-th element in matrix A
α_A, β_A	The secret columnar vectors for A
α_B, β_B	The secret columnar vectors for B
$\widetilde{Res}$	The encrypted result computed by the server
Res	The original result computed by the client
C_1	The result of offline phase
C_2	The result of online phase computed by the server
π	The proof of the returned result
sk	The secret key for the client
h	The hash function
PK	Public key generated by the client
K_{cv}	The common key between the client and the verifier
$[n_1]$	$\{1, 2, \ldots, n_1\}$
n	The size of matrices
Z_m	$\{0, 1, 2, \ldots, m-1\}$
$e(g, h)$	Bilinear pairing

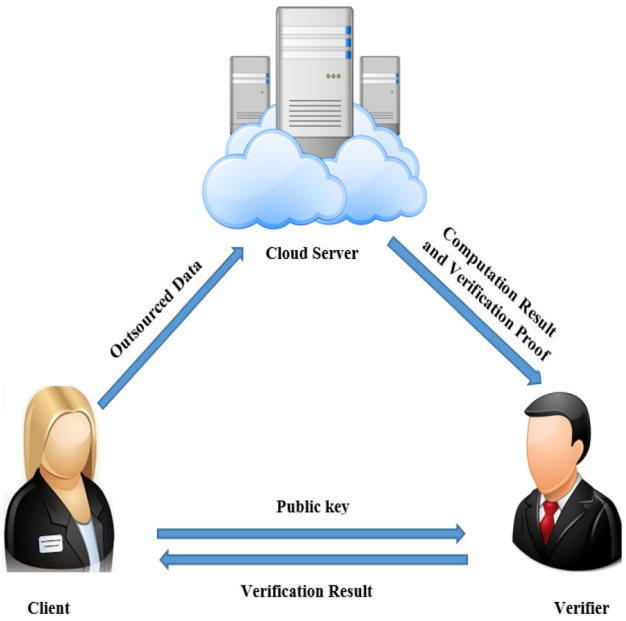

Fig. 1 System Model

- *Non-degeneracy*: for any $g \in G_1$, if for all $h \in G_2$ equation $e(g, h) = 1$ is true then $g = 0$. Moreover, for any $h \in G_2$, there exists some $g \in G_1$ such that $e(g, h) \neq 1$.
- *Efficiency*: there exists an efficient algorithm for computing e.

3.2 System model

As shown in Fig. 1 our proposed scheme has three entities: a computationally weak client, a powerful cloud server and a verifier.

Client: the client wants to outsource matrix multiplication of two large matrices with compatible sizes to a powerful cloud server for decreasing its computational overhead. In order to protect the privacy of data, the client encrypts and sends data to the cloud server. The client performs computations on encrypted matrices to obtain the encrypted result. After receiving encrypted result that is verified by the verifier, the client accepts and decrypts it. On the whole, in our scheme the client can choose some verifiers to verify the correctness of the result. Particularly, public verifiability means that everyone can verify the correctness of the result. It helps the client to be sure about the correctness of the result.

Cloud server: The computational resource-limited client outsources its heavy computations to the cloud server with unlimited computing resources. In our scheme, the cloud server computes multiplication result in encrypted form and a non-interactive proof. Then, it sends the non-interactive proof to the verifier. It is worthwhile to know that one of the reasons that the client outsources its matrices and verify the result by utilizing public verifiability is that some servers may be reluctant to report their failures in order to increase client retention for their own services.

Verifier: the verifier is the third party who can check validity of the result. It receives the encrypted result and the proof from the cloud server and verifies them by using the public keys. If the results are true, the verifier sends them to the client. Now the client is sure about validity of the result.

3.3 Threat model

Generally, there are two types of threat models in outsourcing: semi-honest cloud and malicious cloud. The semi-honest cloud follows the protocol correctly, but it tries to learn additional information about input and output data. While, the malicious cloud can arbitrarily deviate from protocol specification [19]. The malicious cloud server may send random result to the client for saving its resources. In this paper, we assume the cloud server treats as a semi-honest cloud server.

3.4 Design goals

In this part, we define our goals as follows.

Privacy protection of outsourced data: the privacy of outsourced matrices must be protected. In this scheme, we use secret keys to protect privacy of matrices.

Privacy protection of multiplication result: our proposed scheme provides protection of multiplication result by secret matrices.

Unforgeability of proof: We have to guarantee that the cloud server cannot forge the proof.

Arbitrary large matrix: Most of existing works are not flexible enough since one of matrices involved in the multiplication should be fixed. The proposed scheme enables a flexible way to outsource multiplication of any two arbitrary matrices.

Efficiency: In this paper, the proposed scheme will be executed in online mode and does not need any offline preprocessing phase. Therefore, the client tolerates less overhead in total.

Public verifiability: Any verifier can check validity of the result by checking the correctness of the proof. Therefore the client can outsource even the verification of the result.

4 Review of the Zhang-Lei's scheme

In this section, we first describe Zhang-Lei's scheme [30] which contains five phases namely, Encryption, Requesting, Computing, Verification and Decryption phases. In this scheme, the matrices are categorized into two groups as $\mathcal{M}_{A_{n_1\times n_2}}$ and $\mathcal{M}_{B_{n_2\times n_3}}$ beforehand, where each matrix has a unique identifier.

Encryption phase In this phase, the client encrypts matrices $A \in \mathcal{M}_A$ as $\tilde{A} = A + \alpha_A \beta_A^T$ and $B \in \mathcal{M}_B$ as $\tilde{B} = B + \alpha_B \beta_B^T$, where $\alpha_A \in Z_q^{n_1}$, $\beta_A \in Z_q^{n_1}$, $\alpha_B \in Z_q^{n_2}$ and $\beta_B \in Z_q^{n_2}$ are columnar vectors. It computes $W_{n_1\times n_2}$, where $W_{i,j} = g_1^{r_i \tilde{a}_{i,j}}$ and $pk2_i = g_2^{c_i}$, where $r_i = H(ID_A \| i \| SK)$ and $c_j = H(ID_B \| j \| SK)$ and sends them to the cloud server.

Requesting phase In this phase, the client computes $pkt_{i,j} = g_T^{r_i c_j}$, where $g_T = e(g_1, g_2)$ and sends it with identifiers of matrices to the cloud server.

Computing phase The cloud server computes $\widetilde{Res} = \tilde{A} \times \tilde{B}$ and a non-interactive proof $\pi_{n_1\times n_3}$, where its (i, j)-th element is computed as $\pi_{i,j} = \prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{k,j}}$. Then it sends the (encrypted result, proof) pair $(\widetilde{Res}, \pi)$ to the verifier.

Verification phase The verifier judges the correctness of equation for each element of $\widetilde{Res}$ as $e(\pi_{i,j}, pk2_j) = (pkt_{i,j})^{\widetilde{Res}_{i,j}}$. If all the conditions are true, it sends the encrypted result $\widetilde{Res}$ to the client.

Decryption phase After that, the client computes $R = (A\alpha_B)\beta_B^T + \alpha_A(\beta_A^T B) + \alpha_A(\beta_A^T \alpha_B)\beta_B^T$ and obtains the final result as $Res = \widetilde{Res} - R$.

4.1 Forgery attack against Zhang-Lei's scheme

In this section, we find a security vulnerability in Zhang-Lei's scheme and in the next section, we present a new secure and efficient publicly verifiable scheme for outsourcing multiplication of large matrices which fixes this vulnerability. If the cloud server behaves dishonestly, it can pass the verification equation $e(\pi_{i,j}, pk2_j) = (pkt_{i,j})^{\widetilde{Res}_{i,j}}$. More precisely, the attack procedure is as follows.

1. The client and the cloud server proceed the Zhang-Lei's protocol until the cloud server computes the true values for the encrypted result and the proof as $(\widetilde{Res}, \pi)$.
2. Then it chooses an arbitrary scalar $x \in \mathbb{Z}_q^*$.
3. It computes $x.\widetilde{Res}$ as the manipulated encrypted result and tunes the corresponding proof as π^x.
4. Finally, it sends $(\widetilde{Res}', \pi') = (x.\widetilde{Res}, \pi^x)$ as a new (encrypted result and proof) to the verifier. Now we show that this incorrect pair of $(x.\widetilde{Res}, \pi^x)$ passes the verification equation $e(\pi'_{i,j}, pk2_j) = (pkt_{i,j})^{\widetilde{Res}'_{i,j}}$.

Since $\pi_{i,j} = \prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{k,j}}$, $W_{i,k} = g_1^{r_i \tilde{a}_{i,k}}$, $pkt_{i,j} = g_T^{r_i c_j}$ and $pk2_j = g_2^{c_j}$ we have that

$$
\begin{aligned}
e(\pi'_{i,j}, pk2_j) &= e\left(\pi_{i,j}^x, pk2_j\right) \\
&= e\left(\left(\prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{k,j}}\right)^x, pk2_j\right) \\
&= e\left(\left(\prod_{k=1}^{n_2} (g_1^{r_i \tilde{a}_{i,k}})^{\tilde{b}_{k,j}}\right)^x, g_2^{c_j}\right) \\
&= e\left(\left(g_1^{\sum_{k=1}^{n_2} r_i \tilde{a}_{i,k}.\tilde{b}_{k,j}}\right)^x, g_2^{c_j}\right) \\
&= e(g_1, g_2)^{r_i.c_j.x\sum_{k=1}^{n_2} \tilde{a}_{i,k}.\tilde{b}_{k,j}} \\
&= (g_T)^{r_i.c_j.x\sum_{k=1}^{n_2} \tilde{a}_{i,k}.\tilde{b}_{k,j}} \\
&= \left(pkt_{i,j}\right)^{x\sum_{k=1}^{n_2} \tilde{a}_{i,k}.\tilde{b}_{k,j}} \\
&= \left(pkt_{i,j}\right)^{x.\widetilde{Res}} = \left(pkt_{i,j}\right)^{\widetilde{Res}'}
\end{aligned}
$$

The last equation is true because $\sum_{k=1}^{n_2} \tilde{a}_{i,k}.\tilde{b}_{k,j}$ is the (i, j)-th element of $\widetilde{Res} = \tilde{A} \times \tilde{B}$. Thus, the verifier accepts $\widetilde{Res}'$ as the correct encrypted result and sends it to the client. Then the client decrypts $\widetilde{Res}'$ as $Res' = \widetilde{Res}' - R$ which is

not equal to $A \times B$. But the client accepts it as the result $A \times B$. Therefore, this is a successful forgery attack.

Besides, this security vulnerability, the Zhang-Lei's scheme has a preprocessing phase in offline mode. So the client tolerates heavy computational overhead. Also the verifier has to compute n^2 pairings in the verification phase. Therefore this protocol requires heavy computations that may degrade the efficiency of the outsourcing process. Moreover, their scheme is not flexible in the sense that the client is only allowed to request of the multiplication of matrices who has already committed to beforehand. In other words, the client cannot outsource the multiplication of two freshly generated matrices. So we are motivated to enhance Zhang -Lei's scheme in order to make it work for freshly generated matrices, fix the mentioned security vulnerability and finally increase efficiency and computing speed. Specifically, we remove the offline phase and matrix identifiers. In addition, the computational overhead of the verifier side would be reduced into $2n^2$ exponentiations.

5 The proposed scheme

We assume a client outsources two encrypted matrices to the cloud server. The cloud server computes matrix multiplication and also provides a proof π to the verifier. The verifier checks the proof. If the correctness of the result is proved, then the verifier computes *MAC* of the result. Then it sends $\widetilde{Res}$ and *MAC* to the client. Finally, the client checks the validity of *MAC*. If*MAC* is validated, then he accepts the result and decrypts it. It is important to say that the client and the verifier have a common key to compute *MAC*.

In this section, we explain the proposed scheme which consists of five phases: the first phase is **Setup**. The second phase is **Encryption** phase that is executed by the client. Then the **Computing** phase is done by the server. The **Verification** phase is executed by the verifier and finally the client executes the **Decryption** phase.

Setup phase The client chooses two large primes p and q such that $q|p-1$. Then it generates a subgroup G of $\mathbb{Z}_p^*$ with the generator g of order q. Then it chooses a private key $sk \in G$ and a hash function $H : [n_1] \times G \rightarrow \mathbb{Z}_q^*$. After that, the client publishes parameters as $param = (q, g, h)$ which is used by the whole outsourcing system.

Encryption phase in this phase, the client processes matrices before sending them to the cloud server. First the client chooses vectors $\alpha_A \in Z_q^{n_1}$, $\beta_A \in Z_q^{n_2}$, $\alpha_B \in Z_q^{n_2}$ and $\beta_B \in Z_q^{n_3}$ Then it encrypts matrices A and B into $\tilde{A} = A + \alpha_A \beta_A^T$ and $\tilde{B} = B + \alpha_B \beta_B^T$, respectively. After that it sends $\tilde{A}$ and $\tilde{B}$ to the cloud server. Also the client computes matrix $W_{n_1 \times n_2}$, where $W_{i,j} = g^{r_i \tilde{a}_{ij}}$ and $r_i = H(i||sk)$ and sk denotes the client's secret key. Then it computes the public key as $PK_{n_1 \times n_2} = (pk_{i,j}) = (g^{r_i})$ and sends it to the verifier. The details are shown in Algorithm 1.

Algorithm 1 Encryption Phase by the Client

1. **Input:** Matrix A, B
2. **Output:** C_1
3. Select $\alpha_A \in_R Z_q^{n_1}, \alpha_B, \beta_A \in_R Z_q^{n_2}, \beta_B \in_R Z_q^{n_3}$
4. Compute $\tilde{A} = A + \alpha_A \beta_A^T$ and $\tilde{B} = B + \alpha_B \beta_B^T$
5. **for** $i = 1$ *to* n_1 **do**
6. Compute $r_i = h(i||sk)$
7. **for** $j = 1$ *to* n_3 **do**
8. Compute and save $PK = (PK_{i,j})_{n_1 \times n_3} = (g^{r_i})$
9. Compute $W = (W_{i,j})_{n_1 \times n_2} = (g^{r_i \tilde{a}_{i,j}})_{n_1 \times n_2}$
10. **end for**
11. **end for**
12. **return** $C_1 = (\tilde{A}, \tilde{B}, W, PK)$

Computing phase the cloud server receives matrices $\tilde{A}$ and $\tilde{B}$ and computes matrix multiplication result as $\widetilde{Res} = \tilde{A} \times \tilde{B}$ and a non-interactive proof as $\pi = (\pi_{i,j}) = (\prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{kj}})$ where $1 \leq i \leq n_1$, $1 \leq j \leq n_3$. Then it sends $(\widetilde{Res}, \pi)$ to the verifier. The details are shown in Algorithm 2.

Algorithm 2 Computing Phase by the Cloud Server

1. **Input:** $\tilde{A}, \tilde{B}$ and W
2. **Output:** C_2
3. Compute $\widetilde{Res} = \tilde{A} \times \tilde{B}$
4. **for** $i = 1$ *to* n_1 **do**
5. **for** $j = 1$ *to* n_3 **do**
6. **for** $k = 1$ *to* n_2 **do**
7. Compute $\pi = \pi_{i,j_{n_1 \times n_3}} = (\prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{k,j}})_{n_1 \times n_3}$
8. **end for**
9. **end for**
10. **end for**
11. **return** $C_2 = (\widetilde{Res}, \pi)$

Algorithm 3 Verification Phase by the Verifier

1. **Input:** $C_2, z \in \{0, 1\}$
2. **Output:** C_3 *or* $\perp$
3. **for** $i = 1$ *to* n_1 **do**
4. **for** $j = 1$ *to* n_3 **do**
5. **if** $(\pi_{i,j} = (pk_{i,j})^{\widetilde{res}_{i,j}}$ and $\pi_{i,j} = \prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{k,j}})$ **then**
6. $z = 1$
7. **continue;**
8. **else**
9. $z = 0$
10. **return** $\perp$
11. **end if**
12. **end for**
13. **end for**
14. **if** $z = 1$
15. **return** $C_3 = (MAC_{K_{cv}}(PK||\widetilde{Res}), \widetilde{Res})$

Algorithm 4 Decryption Phase by the Client

1. **Input:** C_3
2. **Output:** Res *or* $\perp$
3. **if** MAC is validated **then**
4. Compute $R = (AA\alpha_B)\beta_B^T + \alpha_A(\beta_A^T B) + \alpha_A(\beta_A^T \alpha_B)\beta_B^T$
5. Compute $Res = \widetilde{Res} - R$
6. **return** Res
7. **else**
8. **return** $\perp$
9. **end if**

Verification phase according to the Algorithm 3, this phase is executed by the verifier. After receiving $\widetilde{Res}$ and π, the verifier checks if $\pi_{i,j} = (pk_{i,j})^{\widetilde{res}_{i,j}}$ and $\pi_{i,j} = \prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{k,j}}$.

If these equations hold, it generates MAC of $(\widetilde{Res}, \pi)$ as $MAC_{K_{cv}}(PK, \widetilde{Res})$ by the common key K_{cv} previously shared between the client and the verifier. The verifier sends the MAC with $\widetilde{Res}$ to the client.

Decryption phase as shown in Algorithm 4 to ensure the integrity of the received messages , the client checks the MAC. If the MAC is confirmed , it computes matrix R as $R = (A\alpha_B)\beta_B^T + \alpha_A(\beta_A^T B) + \alpha_A(\beta_A^T \alpha_B)\beta_B^T$ and decrypts $\widetilde{Res}$ as $Res = \widetilde{Res} - R$. Otherwise the client rejects it and sends $\perp$.

6 Security and performance evaluation

In this section first, we prove the correctness of verification and decryption phases. Second, we evaluate the performance of our scheme and compare it with the related work in terms of functionality as well as computation, communication and storage.

6.1 Correctness and security analysis

We prove the correctness of verification and decryption phases and guarantee the security of our scheme as follows.

Correctness guarantee in this scheme, the verifier checks the equations $\pi_{i,j} = (pk_{i,j})^{\widetilde{res}_{i,j}}$ and

$\pi_{i,j} = \prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{k,j}}$. If the computation is performed correctly, the above equations hold. First according to the equation $W_{i,j} = g^{r_i \tilde{a}_{i,j}}$, we replace $W_{i,j}$ by $g^{r_i \tilde{a}_{i,j}}$.

$$\begin{aligned}\pi_{i,j} &= \prod_{k=1}^{n_2} (W_{i,k})^{\tilde{b}_{k,j}} \\ &= \prod_{k=1}^{n_2} (g^{r_i \tilde{a}_{i,k}})^{\tilde{b}_{k,j}} \\ &= \prod_{k=1}^{n_2} (g)^{r_i \tilde{a}_{i,k} \tilde{b}_{k,j}}\end{aligned}$$

Now according to the equation $\sum_{k=1}^{n_2} \tilde{a}_{i,k}\tilde{b}_{k,j} = \widetilde{Res}_{i,j}$, we have

$$\begin{aligned}\pi_{i,j} &= g^{r_i \sum_{k=1}^{n_2} \tilde{a}_{i,k}\tilde{b}_{k,j}} \\ &= (g^{r_i})^{\widetilde{Res}_{i,j}}\end{aligned}$$

Finally, according to the equation $W_{i,k} = g^{r_i \tilde{a}_{i,k}}$, the above equation is written as:

$$\begin{aligned}\pi_{i,j} &= \prod_{k=1}^{n_2} (g^{r_i \tilde{a}_{i,k}})^{\tilde{b}_{k,j}} \\ &= \prod_{k=1}^{n_2} (W_{i,k})^{\tilde{b}_{k,j}}\end{aligned}$$

Now, we evaluate correctness of decryption phase as stated below

$$\begin{aligned}\widetilde{Res} - R = \tilde{A} \times \tilde{B} - R &= (A + \alpha_A \beta_A^T)(B + \alpha_B \beta_B^T) - R \\ &= AB + \underbrace{A\alpha_B\beta_B^T + \alpha_A\beta_A^T B + \alpha_A\beta_A^T\alpha_B\beta_B^T}_{\mathrm{R}} - R \\ &= AB\end{aligned}$$

Security guarantee as mentioned before, the client outsources multiplication of two arbitrary matrices A and B to the cloud server. Actually, it has to send its sensitive data to the untrusted entity. The semi-honest cloud server may send incorrect results to the client. To ensure that the protocol is secure, we analyze security properties of our scheme. The security properties that we focus on are described as follows.

Privacy Protection of Outsourced Data: We demonstrate that our scheme achieves the privacy protection of outsourced data. We assume a semi-honest cloud server tries to learn more information about matrices. To protect of revealing outsourced data, the matrices are encrypted. In this way, the client encrypts matrices A and B as $\tilde{A} = A + \alpha_A \beta_A^T$ and $\tilde{B} = B + \alpha_B \beta_B^T$, respectively. To achieve matrices A and B, the cloud server has to know vectors $\alpha_A, \beta_A^T, \alpha_B$ and β_B^T . These vectors are randomly chosen and kept secret from the cloud server. Therefore, the private information about outsourced matrices could not be obtained by the cloud server and our scheme achieves privacy protection of outsourced matrices.

Privacy protection of result One of the most important properties of outsourced computing is the privacy protection of the result. The cloud server needs matrix R to achieve the decrypted result Res. But matrix R is private and created by the client and the cloud server cannot learn any more information about the original result Res. So the privacy protection of result is achieved.

Unforgeability of result When the verifier sends the result to the client, the attacker can manipulate the result

$\widetilde{Res}$ and sends it to the client. To prevent this attack, the verifier computes MAC of public key and result as $MAC_{K_{cv}}(PK, \widetilde{Res})||\widetilde{Res}$ that receives from the cloud server. The verifier sends MAC to the client and the client checks the validity of the MAC.

Unforgeability of proof Based on the threat model of this scheme, the cloud server may send incorrect results to the client. So the client has to check the correctness of result. It delegates verifying phase to the verifier. The cloud server sends the encrypted result $\widetilde{Res}$ and the proof π to the verifier. The cloud server needs $r_i = H(i||sk)$ to forge the proof and pass the verifier filter. But, based on hardness of the discrete logarithm problem (DLP), the cloud server cannot achieve r_i in the equation $PK = (pk_{i,j}) = (g^{r_i})$. Also it cannot achieve r_i by using equation $W = (W_{i,j}) = (g^{r_i \tilde{a}_{i,j}})$ and having encrypted matrix $\tilde{A}$. On the other hand, due to the one-way hash function H, the cloud server also cannot achieve any information about $r_i = H(i||sk)$ and private key sk. If the cloud server sends $\widetilde{Res}$ and π randomly, it can pass the first equation. To prevent this attack, the verifier has to check the second equation $\pi_{i,j} = \prod_{k=1}^{n_2} W_{i,k}^{\tilde{b}_{k,j}}$. If the cloud server forges the proof π, it cannot pass the second equation. As a result, our proposed scheme will provide the property of unforgeability of proof.

6.2 Performance evaluation

Now, we evaluate the performance of our scheme and compare it with the related work in terms of functionalities as well as computation, communication and storage. Table 2 presents some notations used in this section.

Functionality we compare functionalities among [9, 10, 15, 21, 30, 33] and our scheme. All schemes except [30] are protected against the forgery attack. Also all schemes except [9, 15] have public verifiability. Protocols [10, 15] do not provide privacy protection of multiplication result. Moreover, protocols [10, 21] do not provide privacy protection of outsourced data and one of the matrices involved in the computing has to be fixed, which makes the scheme unsuitable for many applications. In [15, 30, 33] the client chooses matrices among two categories as M_A and M_B which are created in the preprocessing phase. While in our scheme and [9] the matrices are selected arbitrarily whenever the client wants, and the protocol can just be executed in the online mode. The details are shown in Table 3.

Computation overhead as shown in previous section, [9] does not have public verifiability. So in this section, we compare our protocol with [10, 21, 30, 33] in term of computation overhead. First the computation overhead in the offline phase and second the computation overhead in the online phase will be compared. As shown in Table 4 our scheme does not have any offline phase. While [10, 21, 30, 33] have heavy preprocessing phases. As mentioned in Table 3, N is the number of matrices in categories M_A and M_B. Based on [33] we consider $N = 10^4$. In this section we also compare computation overhead of the verifier. The verifier in the verification phase of [10, 21, 30] has to compute n^2 pairings and n^2 exponentiations and in [33] $2n^2$ pairings and $2n^2$ exponentiations. While the proposed scheme has to compute only $2n^2$ exponentiations. So, not only our scheme fixes the mentioned security vulnerability of [30], but also its computation overhead in client side and verifier side are n exponentiations and n^2 pairings less than [30], respectively. As a result, our scheme is more efficient than related work in term of computation overhead.

To support this inference in reality, we estimate the running time of our protocol on a system with core i5 processor 2.7 GHz and 4.0 GB RAM and compare it with [10, 21, 30, 33]. Our estimation is based on the run times of the cryptographic operations reported in [6, 7, 25]. Let all the matrices be square, where $n_1 = n_2 = n_3 = 10^3$.

To clarify, as mentioned before, we specify time of multiplication, exponentiation, pairing and hash function computation in Table 5 based on [6, 7, 25]. In addition, we

Table 2 Notation for performance evaluation

Symbols	Meanings
M	Time for a multiplication operation over $\mathbb{Z}_p$
E	Time for an exponentiation operation over $\mathbb{G}$
P	Time for a pairing operation over $\mathbb{G} \times \mathbb{G}$
H	Time for computing a hash function
M'	Time for checking a MAC
$\lvert\mathbb{Z}_p\rvert$	The size of a number in $\mathbb{Z}_p$
S_{ID}	The length of a matrix identifier ID
n_1, n_2, n_3	The dimensions of matrix
N	The Number of matrices in the M_A and M_B

Table 3 Comparison of functionality

Properties	[9]	[10]	[15]	[21]	[30]	[33]	Ours
Public verifiability	×	√	×	√	√	√	√
Variable matrix	√	×	√	×	√	√	√
Privacy of input	√	×	√	×	√	√	√
Privacy of output	√	×	×	√	√	√	√
Online mode	√	×	×	×	×	×	√
Security	√	√	√	√	Forgeable	√	√

Table 4 Comparison of computation overhead

Schemes	Computation overhead		
	Offline	Online	
	Client	Client	Verifier
[10]	$3n^2(M+E)$	$3n^2M+4n^2E$	$n^2(P+E)$
[21]	$(n^3+n^2)M+n^2P+(2n^2+n)E$	$(3n^2+n+1)M+E+P$	$n^2(P+E+M)$
[30]	$(3n^2)M+(n^2+n)E+2nH$	$(6n^2+n)M+n^2E+2n^2H$	$n^2(P+E)$
[33]	$N(5n^2+9n)M+N(3n^2+5n)H+N(8n+3)E$	$(2n^2+4n)M+3nE+2nH$	$2n^2(P+E+H+M)$
Ours	–	$(7n^2+3n)M+M'+2n(H+nE)$	$2n^2E$

estimate the run time of protocols presented in [10, 21, 30, 33] and compare them with our scheme in terms of client side and verifier side which is shown in Tables 6 and 7, respectively. It is necessary to mention that all the values reported in these tables are in terms of seconds. Further, the dimensions of matrices are supposed to be 1000 to 6000. As can be seen in Table 7, our scheme is at least 15 times faster than the related work in verification side when dimensions increase.

As shown in Figs. 2 and 3, the running time of our scheme are obviously less than those of previous schemes. Therefore, our scheme is more efficient than others in the client and verifier sides. The most striking feature observed from tables and figures is that our proposed scheme has overwhelmingly decreased verifier's computation overhead which demonstrates a major achievement of our proposed scheme clearly.

Communication overhead In this part, we compare communication overhead of related work with our scheme. Our scheme does not have any offline mode. But [10, 21, 30, 33] have an offline phase where the client tolerates massive communication overheads. As a matter of fact, the communication overhead is measured in terms of the number of elements exchanged between the client and the server. This comparison is shown in Table 8.

Storage overhead in this section, we analyze the storage overhead of the client side. In [30] the client has to save secret key, secret parameters and public keys.

Remarkably, in [10, 33] the client has to keep encryption parameters. Whilst, in our scheme the client only saves the secret key and the secret parameters. In fact, the storage overhead is measured in terms of the number of elements stored in the client side. Overall consideration, the storage overhead of our scheme is acceptable. The details are shown in Table 9.

Table 5 Running time of operations

Operation	Time (μ s)
Pairing	6040
Exponentiation	210
Hash function	0.22
Multiplication	0.13

Table 6 Comparison of client runtime

n	[10]	[21]	[30]	[33]	Ours
1000	751.78	6591	421.83	29,930	420.9
2000	3007.12	26,883	1686.7	86,054	1683.6
3000	6766.02	61,655	3795.12	168,378	3788.1
4000	12,028.48	104,199	6746.7	276,904	6734.4
5000	18,794.5	177,764	10,541.3	411,645	10,522.5
6000	27,064.1	260,660	15,179.3	572,556	15,152.4

Table 7 Comparison of verifier runtime

n	[10]	[21]	[30]	[33]	Ours
1000	6250	6250.2	6250	12,500.7	420
2000	25,000	25,000.5	25,000	50,002.8	1680
3000	56,250	56,251.1	56,250	112,506	3780
4000	100,000	100,002	100,000	200,011	6720
5000	156,250	156,253	156,250	312,517	10,500
6000	225,000	225,005	225,000	450,025	15,120

7 Conclusion

In this paper, we presented a secure and efficient protocol for publicly verifiable outsourcing of large matrix multiplication. Our proposed scheme provides a more secure and efficient way for the client to compute multiplication of any two arbitrary matrices in an online mode with public verifiability. More particularly, security analysis

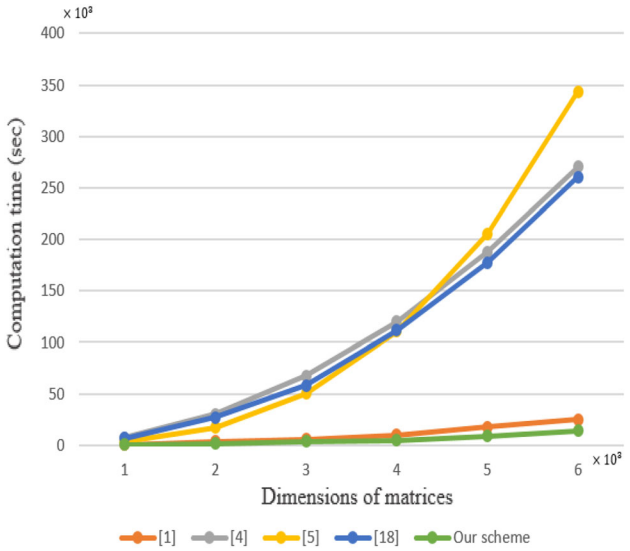

Fig. 2 Comparison of client side overhead

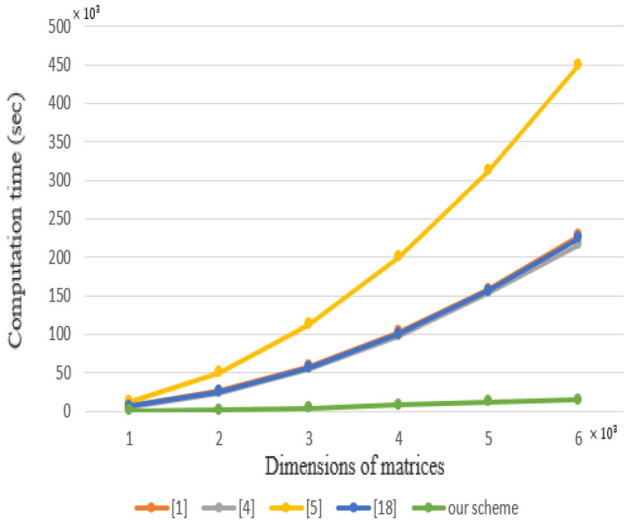

Fig. 3 Comparison of verifier side overhead

Table 8 Comparison of communication overhead

Schemes	Client to server	
	Offline	Online
[10]	$n^2\|\mathbb{Z}_p\| + n^2\|\mathbb{G}\|$	$n^2\|\mathbb{Z}_p\|$
[21]	$n^2\|\mathbb{Z}_p\| + n^2\|\mathbb{G}\|$	$n^3\|\mathbb{Z}_p\|$
[30]	$(2n^2+2)\|\mathbb{Z}_p\|$	$2S_{ID}$
[33]	$(2n^2+7n)\|\mathbb{Z}_p\|$	$2S_{ID}$
Ours	–	$3n^2\|\mathbb{Z}_p\|$

demonstrates that our scheme achieves privacy protection of outsourced data, privacy protection of multiplication result, unforgeability of proof and public verification of the result in online mode. We compared our scheme with related work in terms of functionality, computation, communication and storage overhead. In consequence, the runtime of our protocol has dramatically plummeted due to declining computation overhead of verification side and eliminating offline phase in the client side. Hence, this scheme has a lighter computation, communication and storage overhead than previous works.

Table 9 Comparison of storage overhead

[10]	$(n^2+n+2)\|\mathbb{G}_2\| + 2n\|\mathbb{Z}_p\|$
[21]	$(n^3+3n^2+1)\|\mathbb{Z}_p\|$
[30]	$(n^4+n^2+4n+1)\|\mathbb{Z}_p\|$
[33]	$(20n^2+10n+9)\|\mathbb{Z}_p\|$
Ours	$(3n^2+4n+1)\|\mathbb{Z}_p\|$

References

1. Atallah, M., et al.: Secure outsourcing of scientific computations. Adv. Comput. **54**, 215–272 (2002)
2. Atallah, M., Frikken, K.: Securely outsourcing linear algebra computations. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, pp. 48–59 (2010)
3. Benjamin, D., Atallah, M.: Private and cheating-free outsourcing of algebraic computations. In: PST'08. Sixth Annual Conference on Privacy, Security and Trust, 2008, pp. 240–245 (2008)
4. Chen, Z., et al.: Secure and verifiable outsourcing of large-scale matrix inversion without precondition in cloud computing. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2018)
5. Chen, X., et al.: Efficient algorithms for secure outsourcing of bilinear pairings. Theor. Comput. Sci. **562**, 112–121 (2015)
6. Daly, A., Marnane, W.: Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic. In: Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays, pp. 40–49. ACM (2002)
7. De Caro, A., Iovino, V.: jPBC: Java pairing based cryptography. In: 2011 IEEE Symposium on Computers and Communications (ISCC), pp. 850–855. IEEE (2011)
8. Elkhiyaoui, K. et al.: Efficient techniques for publicly verifiable delegation of computation. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pp. 119–128 (2016)
9. Erfan, F., Mala. H.: Online privacy preserving outsourcing of large matrix multiplication. In: 7th International Conference on Computer and Knowledge Engineering (ICCKE), pp. 235–240, IEEE (2017)
10. Fiore, D., Gennaro, R.: Publicly verifiable delegation of large polynomials and matrix computations, with applications. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 501–512 (2012)
11. Gennaro, R., et al.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Advances in Cryptology-CRYPTO, pp. 465–482. Springer, Berlin (2010)
12. Gentry, C., et al.: Fully homomorphic encryption using ideal lattices. STOC **9**, 169–178 (2009)

13. Hen, X., et al.: New algorithms for secure outsourcing of modular exponentiations. IEEE Trans. Parallel Distrib. Syst. **25**(9), 2386–2396 (2013)
14. Hu, C. et al.: A secure and verifiable outsourcing scheme for matrix inverse computation. In: IEEE INFOCOM 2017-IEEE Conference on Computer Communications pp. 1–9. IEEE (2017)
15. Jia, K., et al.: Enabling efficient and secure outsourcing of large matrix multiplications. In: Conference on IEEE Global Communication (GLOBECOM), pp. 1–6. San Diego, California (2015)
16. Jiang, X., et al.: Secure outsourced matrix computation and application to neural networks. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1209–1222. ACM (2018)
17. Kong, S., et al.: Cloud outsourcing computing security protocol of matrix multiplication computation based on similarity transformation. Int. J. Wirel. Mob. Comput. **14**(1), 90–96 (2018)
18. Lei, X., et al.: Outsourcing large matrix inversion computation to a public cloud. IEEE Trans. Cloud Comput. **1**(1), 1–1 (2013)
19. Lei, X. et al.: Achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud. In: information Science, vol. 280, pp. 205–217 (2014)
20. Lei, X., Liao, X., Huang, T., Li, H.: Cloud computing service: the case of large matrix determinant computation. IEEE Trans. Serv. Comput. **8**(5), 688–700 (2014)
21. Li, H., et al.: Enabling efficient publicly verifiable outsourcing computation for matrix multiplication. In: International Telecommunication Networks and Applications Conference (ITNAC), pp. 44–50. IEEE (2015)
22. Liu, M., et al.: Verifiable outsourcing computation for modular exponentiation from shareable functions. Clust. Comput. (2019). https://doi.org/10.1007/s10586-019-02930-4
23. Ma, X., et al.: Outsourcing computation of modular exponentiations in cloud computing. Clust. Comput. **16**(4), 787–796 (2013)
24. Mohassel, P.: Efficient and Secure Delegation of Linear Algebra. IACR Cryptology ePrint Archive, Report 605 (2011)
25. Speed benchmarks for some of cryptographic algorithms. https://www.cryptopp.com/benchmarks.html
26. Xiao, X., et al.: Efficient and secure outsourcing of DFT, IDFT, and circular convolution. IEEE Access **7**, 60126–60133 (2019)
27. Yao, A.C.: Protocols for secure computations. In: SFCS'08. 23rd Annual Symposium on Foundations of Computer Science, 1982, pp. 160–164. IEEE (1982)
28. Ye, J., et al.: Secure outsourcing of modular exponentiations in cloud and cluster computing. Clust. Comput. **19**(2), 811–820 (2016)
29. Zhang, L.F., Safavi-Naini, R.: 'Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps. International Conference on Cryptology and Network Security, pp. 329–348. Springer, Cham (2013)
30. Zhang, S., et al.: Efficient secure outsourcing computation of matrix multiplication in cloud computing. In: Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2016)
31. Zihao, Sh, et al.: Practical secure computation outsourcing: a survey. ACM Comput. Surv. **51**(2), 31–71 (2019)
32. Zhang, Y., Blanton, M.: Efficient secure and verifiable outsourcing of matrix multiplications. In: International Conference on Information Security, pp. 158–178 (2014)
33. Zhang, S., et al.: Verifiable outsourcing computation for matrix multiplication with improved efficiency and applicability. IEEE Internet Things J. **5**(6), 5076–5088 (2018)

Fatemeh Erfan received her B.S. degree in Computer Engineering from Isfahan University of Technology (IUT) in 2014. She received her M.S. degree in Information Security Engineering from University of Isfahan (UI) in 2018. Her main research interests are in the area of big data, network security, cloud computing and cryptographic protocols.

Hamid Mala received his B.S., M.S. and Ph.D. degrees in Electrical Engineering from Isfahan University of Technology (IUT) in 2003, 2006 and 2011, respectively. He joined University of Isfahan (UI) in September 2011 as an Assistant Professor in the Department of Information Technology Engineering. Currently, he is with the Faculty of Computer Engineering at UI as an Associate Professor. His Research interests include design and cryptanalysis of block ciphers, cryptographic protocols and secure multiparty computation.