sensors

Article

Secure Outsourcing of Matrix Determinant Computation under the Malicious Cloud

Mingyang Song and Yingpeng Sang *

School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China; songmy5@mail2.sysu.edu.cn
* Correspondence: sangyp@mail.sysu.edu.cn

Abstract: Computing the determinant of large matrix is a time-consuming task, which is appearing more and more widely in science and engineering problems in the era of big data. Fortunately, cloud computing can provide large storage and computation resources, and thus, act as an ideal platform to complete computation outsourced from resource-constrained devices. However, cloud computing also causes security issues. For example, the curious cloud may spy on user privacy through outsourced data. The malicious cloud violating computing scripts, as well as cloud hardware failure, will lead to incorrect results. Therefore, we propose a secure outsourcing algorithm to compute the determinant of large matrix under the malicious cloud mode in this paper. The algorithm protects the privacy of the original matrix by applying row/column permutation and other transformations to the matrix. To resist malicious cheating on the computation tasks, a new verification method is utilized in our algorithm. Unlike previous algorithms that require multiple rounds of verification, our verification requires only one round without trading off the cheating detectability, which greatly reduces the local computation burden. Both theoretical and experimental analysis demonstrate that our algorithm achieves a better efficiency on local users than previous ones on various dimensions of matrices, without sacrificing the security requirements in terms of privacy protection and cheating detectability.

Keywords: matrix determinant; secure outsourcing; cloud computing

check for updates

Citation: Song, M.; Sang, Y. Secure Outsourcing of Matrix Determinant Computation under the Malicious Cloud. *Sensors* **2021**, *21*, 6821. https://doi.org/10.3390/s21206821

Academic Editor: Raffaele Gravina

Received: 4 September 2021
Accepted: 8 October 2021
Published: 14 October 2021

1. Introduction

The development of cloud computing provides great convenience to resource-constrained clients. They can outsource complex computations into the cloud by paying a fee. Computation outsourcing brings economic benefits to both resource-constrained clients and high-performance servers. Nevertheless, in practice, cloud servers are untrustworthy, which brings many security issues to computation outsourcing. According to [1], security has become a top issue that affects the business potential of cloud computing. The outsourced data usually contain private user information. The curious cloud may spy on user privacy through outsourced data. Besides, the malicious cloud may violate the computing scripts and return incorrect results to the client. Even without considering the maliciousness of the cloud, computing errors caused by the cloud hardware failure, software errors, etc. should also be detected by the client locally. In addition to these traditional security issues, with the development of smart phones, virtual assistants (VA) have been widely used in smart phones, which are vulnerable to malicious attacks; it uploads voice records to the cloud without the user's knowledge or consent [2]. Therefore, cloud-based secure computation outsourcing algorithms have become hot topics of researches.

There are generally two types of security assumptions in cloud-based secure outsourcing algorithms [3]. The cloud that is assumed to be semihonest (or honest-but-curious) is only curious about the privacy contained in outsourced data. The cloud assumed to be malicious may firstly be semihonest, and then cause damage or forge results to sabotage the computation. Besides, in the above two security assumptions, the local computational burden of the client should be as low as possible. Otherwise, the efficiency benefit of outsourcing will be nullified. Therefore, the secure outsourcing algorithms under the malicious model require the considerations of efficiency, privacy protection, and cheating detectability.

In order to reduce the local computational burden using cloud computation outsourcing while protecting privacy and detecting false results, researchers have proposed secure outsourcing algorithms for many commonly used and complex computations, including the computation of matrix determinant. The computation of matrix determinant has appeared more and more widely in science and engineering problems recently. Many researches of medicine and biology use the matrix determinant for signal processing before using machine learning algorithms for practical classification tasks. For example, when machine learning algorithms are used for medical diagnosis, the data collected by the medical sensors are usually time series and contain private user information. Many researches divide the signal into several segments and fill them into matrices. Then, the determinants of matrices are considered as the feature values of the original signal. However, the computation of matrix determinant is still unaffordable for the sensors. They usually need to outsource matrix determinant computations to high-performance hardware.

To improve the detectability of cheating behaviors from the malicious cloud, all previous algorithms must increase the rounds of verification, which significantly increase the local computational burden of the client. Moreover, the currently known algorithm with the best privacy protection in [4] uses significantly more local computations than other algorithms. Therefore, we aim to propose a novel algorithm for secure outsourcing of matrix determinant computation, to solve the conflicts between the efficiency and security issues. The contributions of this paper are summarized as follows:

- We propose a secure outsourcing algorithm for the matrix determinant computation under the malicious cloud model, which can not only ensure the confidentiality of matrix, but also detect the forged results returned from the malicious cloud. We use the permutation, mix-row/mix-column, and split operations in our algorithm to protect privacy, which achieves the currently known lowest computation cost.
- We propose a one-round verification method in the proposed algorithm, which achieves a high cheating detectability. The malicious forged results can only escape our local verification with the probability of $\frac{1}{(n!)^4}$, given a matrix of $n \times n$ dimensions. In all the previous algorithms, the detectability of malicious forged results depends on the rounds of verification, and to achieve a high cheating detectability, multiple rounds of verification are required, which also brings high computational burden to the client. In the previous three algorithms [4–6], the succeeding probability of malicious forged results is $\frac{1}{2^l}$, where l is the number of verification rounds and recommended to be greater than 20.
- We conduct theoretical proofs of the correctness, efficiency, privacy protection, and cheating detectability for the proposed algorithm. Experimental results also demonstrate the superior efficiency of the proposed algorithm.

The rest of the paper is organized as follows: We introduce the related work and comparative analysis in Section 2. In Section 3, we introduce some background knowledge and the system model of our algorithm. Section 4 describes the proposed secure outsourcing algorithm for matrix determinant. We conduct the correctness, security, and complexity analysis of the proposed algorithm in Section 5. We evaluate the performance of our algorithm in Section 6. Finally, we conclude our work in Section 7.

2. Related Work and Comparative Analysis

Recently, the privacy and security issues in lightweight devices are widely concerned (e.g., the intrusion detection system on lightweight devices [7] and the secure computation outsourcing on resource-constrained devices). As we all know, although cloud computation outsourcing brings convenience to resource-constrained devices, it also causes privacy issues. Therefore, there are many researches on how to protect user privacy and verify the correctness of results when using cloud computing. From the perspective of the cloud, data access control mechanisms can be used to protect user privacy and ensure data availability. Kayes et al. [8] gave a survey on context-aware access control mechanisms (CAAC) during data management in cloud and fog networks. They also proposed a new generation of Fog-Based CAAC (FB-CAAC) framework for accessing data from distributed cloud data centers. When computations on the encrypted data are required, access control mechanisms are not enough. From the perspective of the client, Fully Homomorphic Encryption algorithms (FHE) [9–12] and Attribute-based Encryption algorithms (ABE) [13–15] have great application potential in cloud secure computation outsourcing, but their high computational complexities limit their practical applications, especially for resource-constrained devices. In addition, there are a large number of researches on the secure outsourcing algorithms for commonly-used and complex scientific computations (e.g., modular exponentiation [16,17], extended Euclidean [18], bilinear pairings [19–21], polynomial multiplication [22]).

There are many applications of matrix in the field of computer science, such as Digital Image Processing (DIP), computer graphics, computer geometry, Artificial Intelligence (AI), network communications, and so on. Thus, many computations of matrix are also commonly used and complex. However, some scientific computations of matrix have high computational complexities. Therefore, there are also many researches on secure outsourcing algorithms for computations of matrix. For example, the secure outsourcing algorithm for matrix multiplication has been widely studied [23–26]. In addition, Non-negative Matrix Factorization (NMF) is widely used in DIP, face recognition, text analysis, and other fields. Thus, there are many secure outsourcing algorithms for NMF [27–30]. Matrix inverse is also one of the most basic computations in large-scale data analysis. Computing the inverse of matrix on resource-constrained devices such as sensors is usually costly. Thus, there are also some secure outsourcing algorithms for matrix inverse [31–33]. Besides, in the field of machine learning, Singular Value Decomposition (SVD) has a wide range of applications. It can be used not only for feature decomposition in dimension reduction algorithms but also recommendation system, Natural Language Processing (NLP), and other fields. Securely outsourcing SVD to the cloud can greatly reduce the computation costs of the client. Chakan et al. proposed a secure outsourcing algorithm for SVD [34]. The local computational complexity of this algorithm is $O(n^2)$ and the complexity of cloud is $O(n^3)$. Chen et al. proposed a secure outsourcing scheme for SVD with less interactions between the client and the cloud [35].

Similar to the above computations of matrix, the determinant is also an important computation of matrix in the field of scientific and engineering. In the semihonest model, Kim et al. proposed a secure matrix determinant outsourcing method based on FHE [36]. Their scheme computes the determinant by the standard definition of matrix determinant, which results in a high computational burden of the client. Zong et al. introduced a division-free computational method for FHE-based secure matrix determinant computation outsourcing [37], which is significantly more efficient than the method in [36]. The above two algorithms only considered the privacy under the semihonest model.

However, in practice, the cloud may be malicious. To the best of our knowledge, the existing secure outsourcing algorithms for matrix determinant computation under the malicious model include [4–6]. In [5], Lei et al. used the block matrix and permutation techniques to protect privacy. In their algorithm, the client's local computations include $(2+l)n^2 + 2m^2 + 4mn + 2n + m$ multiplications, where m is the increase in dimension after encryption and n is the original dimension of matrix. Liu et al. proposed a new matrix determinant secure outsourcing algorithm using the permutation and mix-row/mix-column operations, which avoid the increase in matrix dimension during the encryption and reduces the number of local multiplications to $(2+l)n^2 + 3n$ [6]. Zhang et al. proposed a method that has better privacy [4]; however, because $8n$ times of elementary column/row transformations are involved in the process of encryption, it has a higher local computational burden than other algorithms, which require $(10+l)n^2 + 6n$ local multiplications. All the above three algorithms and our proposed algorithm have the same cloud computational complexity ($O(n^{2.373})$).

In our algorithm, in addition to the permutation and mix-row/mix-column operations, we used split operation to achieve a higher privacy. To prevent malicious cloud from forging computation results, the above three algorithms adopted the idea of Freivalds' algorithm [38] with ln^2 computation costs that need to increase the frequency of verification (l) to improve the cheating detectability. l is greater than 20 at least in their works. In our proposed algorithm, only one round of verification is required, so there is no factor l in the computation cost. Table 1 demonstrates comparisons of our algorithm and the three existing algorithms from aspects of local multiplications, privacy protection level, and cheating detectability. Since multiplications dominate the local computation, local additions are omitted here and will be discussed later in Section 5. As we will discuss in Section 5, our proposed algorithm uses the lowest overall local computation cost to achieve a high detectability of result cheating (or a negligible probability of cheating success).

Table 1. Efficiency and Security comparative assessment.

Algorithm	Local Multiplications	Probability of Privacy Leakage	Probability of Cheating Success
Our algorithm	$12n^2 + 12n$	$\frac{1}{(n!)^4(2^\lambda)^{4n}}$	$\frac{1}{(n!)^4}$
Lei's algorithm [5]	$(2+l)n^2 + 2m^2 + 4mn + 2n + m$	$\frac{1}{((n+m)!)^2(2^\lambda)^{2(n+m)}}$	$\frac{1}{(2)^l}$
Liu's algorithm [6]	$(2+l)n^2 + 3n$	$\frac{1}{(n!)^2(2^\lambda)^{2n}}$	$\frac{1}{(2)^l}$
Zhang's algorithm [4]	$(10+l)n^2 + 6n$	$\frac{1}{(n!)^2(2^\lambda)^{10n-8}}$	$\frac{1}{(2)^l}$

3. Preliminary

The notations and their implications used in this paper are shown in Table 2. This section will introduce some background knowledge of our algorithm.

Table 2. Symbols and implications.

Symbol	Implication
$\boldsymbol{r}$	A vector
$\boldsymbol{M}$	A full rank $n \times n$ matrix
$det(\boldsymbol{M})$	The determinant of matrix $\boldsymbol{M}$
$\alpha \leftarrow \mathcal{K}$	Choose an element α from set $\mathcal{K}$ randomly
λ	Security parameter
$Prob_A^x(\chi)$	The probability that attacker A obtains the secret input x using data χ
$Prob_C^{forge}(\chi)$	The probability that the client C detects the forged results $forge$ using data χ
$M(i,j), M_{i,j}$	The element in the ith row and jth column of matrix $\boldsymbol{M}$
$\boldsymbol{m}_{i,-}$ / $\boldsymbol{m}(i,-)$	The ith row of matrix $\boldsymbol{M}$
$\boldsymbol{m}_{-,j}$ / $\boldsymbol{m}(-,j)$	The jth column of matrix $\boldsymbol{M}$
$\boldsymbol{M}^T$	The transpose of $\boldsymbol{M}$
$\boldsymbol{a} \centerdot \boldsymbol{b}$	The inner product of vector $\boldsymbol{a}$ and vector $\boldsymbol{b}$
$f_{LU}(\boldsymbol{M})$	The Low triangle and Up triangle decomposition of matrix $\boldsymbol{M}$

3.1. System Model

The secure outsourcing algorithm for matrix determinant consists of the following five parts.

- KeyGen(λ) $\rightarrow$ (SK_1, SK_2): λ is a security parameter related to key generation. The generated key SK_1 is used to encrypt the input data, and SK_2 is used to verify and decrypt the returned results. Both SK_1 and SK_2 should be kept privately by the client C.

- Encrypt(x, SK_1) $\rightarrow$ (σ_x): x is the input data. The client uses SK_1 to encrypt the input x and gets encrypted data σ_x. σ_x is sent to the server S for computing.
- Compute(f, σ_x) $\rightarrow$ (σ_y): f is a function given by the client. The server computes σ_y using the given function f and encrypted data σ_x.
- Verify(σ_y, SK_2) $\rightarrow$ $(True/\perp)$: The client verifies the results returned from the cloud. If the σ_y is valid, the output of this function is *True*. Otherwise, the output is $\perp$.
- Decrypt(σ_y, SK_2) $\rightarrow$ (y): The client uses the secret key SK_2 to decrypt σ_y and obtains the result y.

The proposed algorithm is effective in the malicious cloud model. The malicious cloud can not only use its known information to infer the privacy of the client, but also maliciously forge false computation results to tamper with the whole algorithm. The system model of secure outsourcing for matrix determinant in this paper is shown as Figure 1.

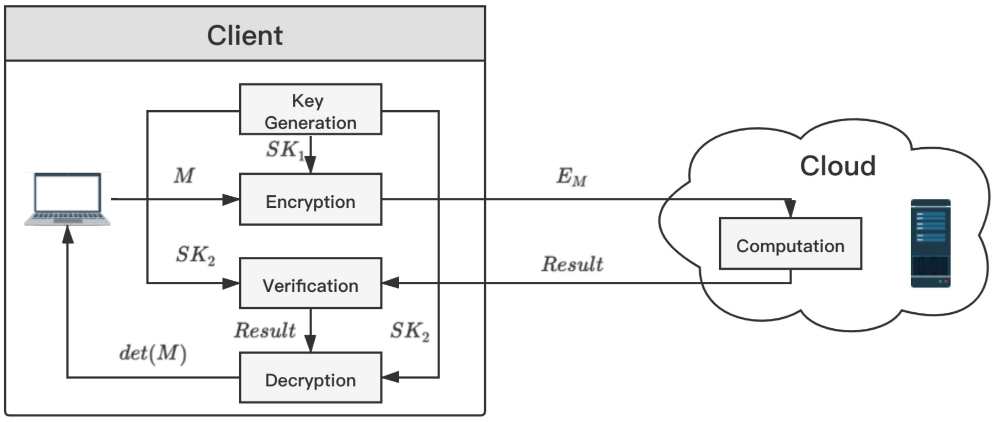

Figure 1. System model of secure outsourcing for matrix determinant computation.

3.2. *Definitions of Correctness, Efficiency, and Security*

When the key generation function $KeyGen(\lambda)$ produces (SK_1, SK_2) satisfying $\forall x \in Domain_{define}$ ($Domain_{define}$ is the domain of input x), if *Encrypt*(x, SK_1) $\rightarrow$ (σ_x) and *Compute*(f, σ_x) $\rightarrow$ (σ_y), then y = *Decrypt*(σ_y,SK_2). The outsourcing algorithm is *correct*.

A semihonest attacker may only be curious about the privacy contained in outsourced data. As shown in Equation (1), for any attacker A in the cloud server, if the probability of computing the secret input x with its known information (σ_x, f, σ_y) is so small that it can be ignored in the polynomial time, the proposed secure outsourcing algorithm for matrix determinant is *privacy-protected*.

$$Prob_A^x(\sigma_x, f, \sigma_y) \rightarrow negli \tag{1}$$

A malicious attacker may cause damage or forge results to sabotage the computation. As shown in Equation (2), for any forged computation results ($forge$) returned from the cloud server, if the probability that the client (C) recognizes the forged results using the *Verify*(σ_y,SK_2) function is infinitely close to 1, the proposed secure outsourcing algorithm for matrix determinant is *cheating-detected*, which means the *cheating detectability* of the algorithm is high, and the success probability of attacker's result cheating is negligible.

$$Prob_C^{forge}(Verify, SK_2, \sigma_y) \rightarrow 1 \tag{2}$$

If the local computation complexity of the client is substantially less than the computation complexity of the previous algorithm without outsourcing, the secure outsourcing algorithm for matrix determinant is *efficient*.

4. Secure Outsourcing of Matrix Determinant

In this section, we propose a secure outsourcing algorithm for matrix determinant. In comparison with the previous algorithms, our algorithm aims to improve the privacy protection ability and cheating detectability with less local computation costs. We use the permutation, mix-row/mix-column, and split operations in our algorithm to protect the privacy of matrix. Besides, we use a new result verification method to improve the cheating detectability, which is different from [4–6]. The main idea of the verification method is to ensure that at least the diagonal elements in the results of Low triangle and Up triangle (LU) decomposition returned from the cloud are correct. The security analysis in Section 5 will prove that it is more difficult for the forged results returned from the malicious cloud to pass this verification than the previous ones. The proposed algorithm is described as follows. The size of the input matrix is $n \times n$ in the rest of this section.

4.1. Key Generation

- The client needs to input a key space $\mathcal{K}$. Eight diagonal matrices $\{\boldsymbol{P_1}, \boldsymbol{P_2}, \boldsymbol{P_3}, \boldsymbol{P_4}, \boldsymbol{Q_1}, \boldsymbol{Q_2}, \boldsymbol{Q_3}, \boldsymbol{Q_4}\}$ are generated by selecting random nonzero values from $\mathcal{K}$. Then, the client picks 8 random parameters $n_1, n_2, n_3, n_4, m_1, m_2, m_3, m_4$ (lines 2 and 3 of Algorithm 1).
- The client computes $SK_2 \leftarrow \{t_1, t_2, t_3, t_4\}$ by lines 4–6 of Algorithm 1. It is easy to see that t_i in SK_2 is the determinant of $\boldsymbol{P_i Q_i}$.
- The client computes SK_1 by lines 7–14 of Algorithm 1.

Algorithm 1 Procedure of Secret Key Generation

```
1:  function KeyGen($\lambda$)
2:      Select random nonzero values from $\mathcal{K}$ to generate 8
              diagonal matrices $\{P_1, P_2, P_3, P_4, Q_1, Q_2,$
              $Q_3, Q_4\}$
3:      Pick 8 random parameters satisfying
              $n \le n_1, n_2, n_3, n_4, m_1, m_2, m_3, m_4 \le 2n$
4:      for $i = 1 \to 4$ do
5:          $t_i \leftarrow (-1)^{n_i+m_i} \prod_{j=1}^{n} P_i(j,j)Q_i(j,j)$
6:      end for
7:      for $i = 1 \to 4$ do
8:          for $j = 1 \to n_i$ do
9:              Randomly select two rows of $P_i$ and exchange them.
10:         end for
11:         for $j = 1 \to m_i$ do
12:             Randomly select two columns of $Q_i$ and exchange them.
13:         end for
14:     end for
15:     return $SK_1 \leftarrow \{P_1, P_2, P_3, P_4, Q_1, Q_2, Q_3, Q_4\}$
               $SK_2 \leftarrow \{t_1, t_2, t_3, t_4\}$
16: end function
```

In the *KeyGen* function, $\mathcal{K}$ can be $\{0,1\}^{\lambda}$, given a security parameter λ. From the analysis in Section 5, the number of elements in $\mathcal{K}$ is associated with the security of the algorithm. When $\lambda = 10$, the probability that the cloud obtains the privacy input is less than $\frac{1}{2^{40}}$, which is negligible. Other ways of defining $\mathcal{K}$ are also applicable, as long as the number of elements in the set $\mathcal{K}$ is sufficient to resist the brute-force attack of the cloud.

$$\boldsymbol{m}_i' + \boldsymbol{m}_i'' = \boldsymbol{m}_i \tag{3}$$

$$\boldsymbol{m}_j' + \boldsymbol{m}_j'' = \boldsymbol{m}_j \tag{4}$$

4.2. Encryption

- While keeping the other rows, the client randomly splits the ith and jth rows of $\boldsymbol{M}$ to get four matrices $\boldsymbol{M_1}, \boldsymbol{M_2}, \boldsymbol{M_3}, \boldsymbol{M_4}$. (lines 2–3 of Algorithm 2).
- The client computes $\sigma_x \leftarrow \{\boldsymbol{Y_1}, \boldsymbol{Y_2}, \boldsymbol{Y_3}, \boldsymbol{Y_4}\}$ by lines 4–7 of Algorithm 2.

Algorithm 2 Procedure of Encryption

```
1: function ENCRYPT($\boldsymbol{M} = [\boldsymbol{m_1}, \ldots, \boldsymbol{m_n}]^T$,$SK_1$)
2:     Pick random $i, j$ $(1 \leq i, j \leq n)$
3:     Construct matrices
           $\boldsymbol{M_1} = [\boldsymbol{m_1}, \ldots \boldsymbol{m'_i} \ldots, \boldsymbol{m_n}]^T$;
           $\boldsymbol{M_2} = [\boldsymbol{m_1}, \ldots \boldsymbol{m''_i} \ldots, \boldsymbol{m_n}]^T$;
           $\boldsymbol{M_3} = [\boldsymbol{m_1}, \ldots \boldsymbol{m'_j} \ldots, \boldsymbol{m_n}]^T$;
           $\boldsymbol{M_4} = [\boldsymbol{m_1}, \ldots \boldsymbol{m''_j} \ldots, \boldsymbol{m_n}]^T$;
           satisfying Equations (3) and (4).
4:     $\boldsymbol{Y_1} \leftarrow \boldsymbol{P_1 M_1 Q_1}$.
5:     $\boldsymbol{Y_2} \leftarrow \boldsymbol{P_2 M_2 Q_2}$.
6:     $\boldsymbol{Y_3} \leftarrow \boldsymbol{P_3 M_3 Q_3}$.
7:     $\boldsymbol{Y_4} \leftarrow \boldsymbol{P_4 M_4 Q_4}$.
8:     return $\sigma_x = \{\boldsymbol{Y_1}, \boldsymbol{Y_2}, \boldsymbol{Y_3}, \boldsymbol{Y_4}\}$
9: end function
```

In the algorithm of *Encrypt*, the computations of matrices $(\boldsymbol{M_1}, \boldsymbol{M_2}, \boldsymbol{M_3}, \boldsymbol{M_4})$ involve the generations of vectors $\boldsymbol{m'_i}$, $\boldsymbol{m''_i}$, $\boldsymbol{m'_j}$, and $\boldsymbol{m''_j}$. We can randomly pick n numbers as the elements of vector $\boldsymbol{m'_i}$ from $\mathcal{K}$. Then, we can compute $\boldsymbol{m''_i} = \boldsymbol{m_i} - \boldsymbol{m'_i}$. In the same way, we can compute $\boldsymbol{m'_j}$ and $\boldsymbol{m''_j}$. After computing the matrices $\boldsymbol{Y_1}, \boldsymbol{Y_2}, \boldsymbol{Y_3}, \boldsymbol{Y_4}$, the client sends the matrices $\boldsymbol{Y_1}, \boldsymbol{Y_2}, \boldsymbol{Y_3}, \boldsymbol{Y_4}$ to the cloud.

4.3. Computations of Server

- After receiving the matrices, the cloud computes the LU decomposition for $\boldsymbol{Y_1}$, $\boldsymbol{Y_2}$, $\boldsymbol{Y_3}$, and $\boldsymbol{Y_4}$ to get $\boldsymbol{L_1}, \boldsymbol{U_1}$, $\boldsymbol{L_2}, \boldsymbol{U_2}, \boldsymbol{L_3}, \boldsymbol{U_3}$, $\boldsymbol{L_4}, \boldsymbol{U_4}$, where f_{LU} is the LU decomposition function (lines 2–5 of Algorithm 3).
- The cloud computes determinants of $\boldsymbol{Y_1}, \boldsymbol{Y_2}, \boldsymbol{Y_3}, \boldsymbol{Y_4}$ and obtains r_1, r_2, r_3, r_4. (lines 6–9 of Algorithm 3).
- The cloud returns $\sigma_y = \{r_1, r_2, r_3, r_4, \boldsymbol{L_1}, \boldsymbol{U_1}, \boldsymbol{L_3}, \boldsymbol{U_3}\}$ to the client.

$$r_1 \overset{?}{=} \prod_{i=1}^{n} L_1(i,i) U_1(i,i) \tag{5}$$

$$r_3 \overset{?}{=} \prod_{i=1}^{n} L_3(i,i) U_3(i,i) \tag{6}$$

$$\frac{r_1}{t_1} + \frac{r_2}{t_2} \overset{?}{=} \frac{r_3}{t_3} + \frac{r_4}{t_4} \tag{7}$$

Algorithm 3 Procedure of Computation

```
1:  function COMPUTE($\sigma_x$,$f_{LU}$)
2:      $\{\boldsymbol{L_1}, \boldsymbol{U_1}\} \leftarrow f_{LU}(\boldsymbol{Y_1})$.
3:      $\{\boldsymbol{L_2}, \boldsymbol{U_2}\} \leftarrow f_{LU}(\boldsymbol{Y_2})$.
4:      $\{\boldsymbol{L_3}, \boldsymbol{U_3}\} \leftarrow f_{LU}(\boldsymbol{Y_3})$.
5:      $\{\boldsymbol{L_4}, \boldsymbol{U_4}\} \leftarrow f_{LU}(\boldsymbol{Y_4})$.
6:      $r_1 \leftarrow \prod_{i=1}^{n} L_1(i,i)U_1(i,i)$.
7:      $r_2 \leftarrow \prod_{i=1}^{n} L_2(i,i)U_2(i,i)$.
8:      $r_3 \leftarrow \prod_{i=1}^{n} L_3(i,i)U_3(i,i)$.
9:      $r_4 \leftarrow \prod_{i=1}^{n} L_4(i,i)U_4(i,i)$.
10:     return $\sigma_y = \{r_1, r_2, r_3, r_4, \boldsymbol{L_1}, \boldsymbol{L_3}, \boldsymbol{U_1}, \boldsymbol{U_3}\}$
11: end function
```

4.4. Verification

- The client initializes the $flag$ to *true*.
- The client performs the verifications of Equations (5)–(7). If any of them are invalid, the function returns $\perp$ (lines 3–5 of Algorithm 4).
- The verifications in lines 6–17 verify all the diagonal elements in $\boldsymbol{L_1}, \boldsymbol{L_3}, \boldsymbol{U_1}, \boldsymbol{U_3}$ at least once. If any verification is invalid, the function returns $\perp$.
- If all the verifications are valid, the function returns *true*.

If the output of the *Verify* function is true, the clients executes the *Decrypt* function. Otherwise, the client rejects the results returned from the cloud.

Algorithm 4 Procedure of Verification

```
1:  function VERIFY($\sigma_y$,$SK_2$)
2:      $flag \leftarrow true$
3:      if any verification of Equations (5)–(7) is invalid then
4:          $flag \leftarrow \perp$; return $flag$.
5:      end if
6:      for $i = 1 \rightarrow n$ do
7:          Pick $j, k$ in $[i, n]$ randomly.
8:          if $Y_1(i,j) \neq \boldsymbol{l_1(i,-)} \cdot \boldsymbol{u_1(-,j)}^T$ or
            $Y_3(i,k) \neq \boldsymbol{l_3(i,-)} \cdot \boldsymbol{u_3(-,k)}^T$ then
9:              $flag \leftarrow \perp$; return $flag$.
10:         end if
11:     end for
12:     for $j = 1 \rightarrow n$ do
13:         Pick $i, k$ in $[1, j]$ randomly.
14:         if $Y_1(i,j) \neq \boldsymbol{l_1(i,-)} \cdot \boldsymbol{u_1(-,j)}^T$ or
            $Y_3(k,j) \neq \boldsymbol{l_3(k,-)} \cdot \boldsymbol{u_3(-,j)}^T$ then
15:             $flag \leftarrow \perp$; return $flag$.
16:         end if
17:     end for
18:     return $flag$
19: end function
```

4.5. Decryption

- The client computes the result (Algorithm 5) $det(\boldsymbol{M}) = y = \frac{r_1}{t_1} + \frac{r_2}{t_2}$.

Algorithm 5 Procedure of Decryption

```
1:  function DECRYPT($\sigma_y$,$SK_2$)
2:      $y \leftarrow \frac{r_1}{t_1} + \frac{r_2}{t_2}$
3:      return $y$
4:  end function
```

Obviously, the input parameters of the *Verify* and *Decrypt* functions are the same. In fact, the result of the *Decrypt* function can be obtained during the execution of the *Verify* function in the proposed algorithm. We separate them into two parts for the sake of clarity and convenience to compare with the previous algorithms. The specific flowchart of the proposed algorithm is shown in Figure 2.

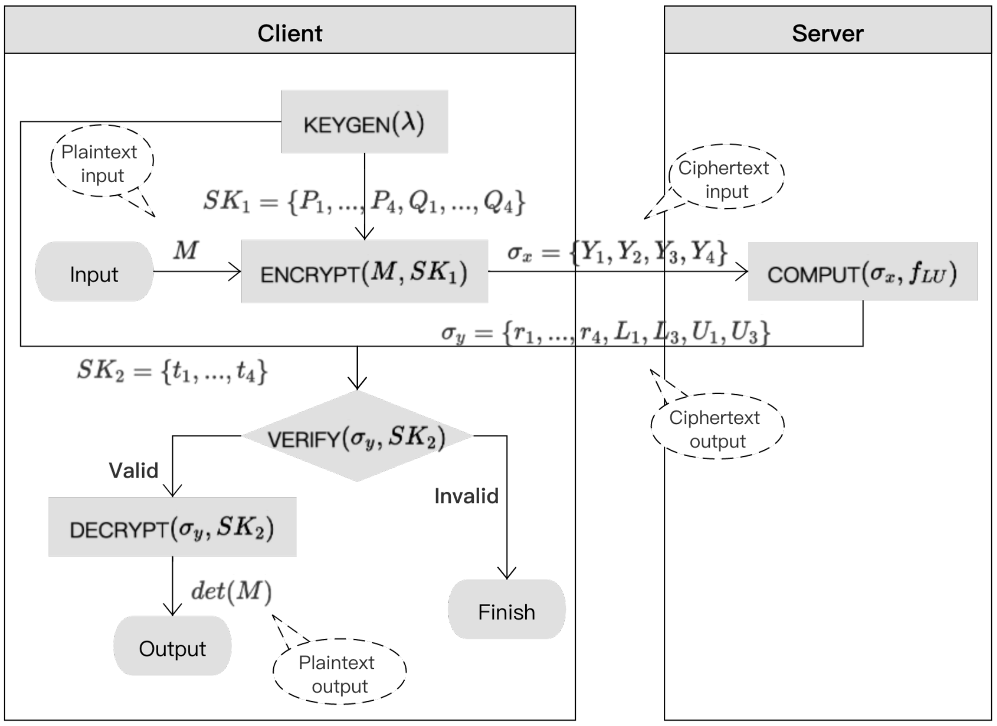
Client
Server
KEYGEN(λ)
Plaintext input
$SK_1 = \{P_1, ..., P_4, Q_1, ..., Q_4\}$
Ciphertext input
Input
M
ENCRYPT(M, SK_1)
$\sigma_x = \{Y_1, Y_2, Y_3, Y_4\}$
COMPUT(σ_x, f_{LU})
$\sigma_y = \{r_1, ..., r_4, L_1, L_3, U_1, U_3\}$
$SK_2 = \{t_1, ..., t_4\}$
Ciphertext output
VERIFY(σ_y, SK_2)
Valid
Invalid
DECRYPT(σ_y, SK_2)
$det(M)$
Finish
Output
Plaintext output

Figure 2. Flowchart of the secure outsourcing algorithm for matrix determinant computation.

In the next section, the correctness, security, and computational complexity of the proposed algorithm will be analyzed.

5. Correctness, Security, and Computational Complexity Analysis

5.1. Correctness

Theorem 1. *The proposed secure outsourcing algorithm for matrix determinant is correct.*

Proof of Theorem 1. When proving the correctness of a secure outsourcing algorithm, it is reasonable to believe that both the client and the cloud honestly follow the procedure of the algorithm. In the procedure of $DECRYPT$, the determinant of matrix is computed by

$$det(\boldsymbol{M}) = DECRYPT(\sigma_y, SK_2) = \frac{r_1}{t_1} + \frac{r_2}{t_2} \tag{8}$$

where

$$r1 = det(\boldsymbol{Y_1}) = det(\boldsymbol{P_1})det(\boldsymbol{M_1})det(\boldsymbol{Q_1}) \tag{9}$$

$$r2 = det(\boldsymbol{Y_2}) = det(\boldsymbol{P_2})det(\boldsymbol{M_2})det(\boldsymbol{Q_2}) \tag{10}$$

According to the function $KEYGEN$, we can obtain

$$t1 = det(\boldsymbol{P_1})det(\boldsymbol{Q_1}) \tag{11}$$

$$t2 = det(\boldsymbol{P_2})det(\boldsymbol{Q_2}) \tag{12}$$

Thus, we can obtain

$$\begin{aligned} det(\boldsymbol{M}) &= det(\boldsymbol{M_1}) + det(\boldsymbol{M_2}) \\ &= \frac{det(\boldsymbol{P_1})det(\boldsymbol{M_1})det(\boldsymbol{Q_1})}{det(\boldsymbol{P_1})det(\boldsymbol{Q_1})} + \frac{det(\boldsymbol{P_2})det(\boldsymbol{M_2})det(\boldsymbol{Q_2})}{det(\boldsymbol{P_2})det(\boldsymbol{Q_2})} \\ &= \frac{det(\boldsymbol{Y_1})}{det(\boldsymbol{P_1})det(\boldsymbol{Q_1})} + \frac{det(\boldsymbol{Y_2})}{det(\boldsymbol{P_2})det(\boldsymbol{Q_2})} \\ &= \frac{r_1}{t_1} + \frac{r_2}{t_2} \end{aligned} \tag{13}$$

Finally, Equation (8) is proved. This implies that the function $DECRYPT$ always yields the correct determinant and the proposed algorithm is correct. □

5.2. Computational Complexity

We analyze the computational complexities of the client and the cloud in this section. The $KEYGEN$ function, $ENCRYPT$ function, $VERIFY$ function, and $DECRYPT$ function are executed by the client. The $COMPUTE$ function is executed by the cloud. We first analyze the computational complexity of the client.

Theorem 2. *The computational complexity of the client side of the proposed secure outsourcing algorithm for matrix determinant is $O(n^2)$.*

Proof of Theorem 2. The operations involved in the $KEYGEN$ function include multiplication and mix-row/mix-column, where multiplications and additions need to be computed. The computations in lines 4–6 require a total of $8n$ multiplications. The computations in lines 7–14 and line 5 require a total of $8n + 4$ multiplications.

The $ENCRYPT$ function involves matrix split operations and matrix multiplications. Obviously, the matrix split operation needs only $2n$ subtractions. The major computations are lines 4–7. Since $\boldsymbol{P_1}, \boldsymbol{P_2}, \boldsymbol{P_3}, \boldsymbol{P_4}, \boldsymbol{Q_1}, \boldsymbol{Q_2}, \boldsymbol{Q_3}$, and $\boldsymbol{Q_4}$ are obtained by randomly swapping rows/columns of diagonal matrices, the computations of $\boldsymbol{Y_1}, \boldsymbol{Y_2}, \boldsymbol{Y_3}$, and $\boldsymbol{Y_4}$ require a total of $8n^2$ multiplications; therefore, the $ENCRYPT$ function requires a total of $8n^2$ multiplications and $2n$ subtractions.

In the $VERIFY$ function, the verifications of Equations (5)–(7) require $4n$ multiplications. The verifications of lines 6–17 compute $4n$ times of the vector's inner product, which require $4n^2$ additions and multiplications. The verification in Equation (7) requires 2 additions. Therefore, the $VERIFY$ function requires $4n^2 + 4n$ multiplications and $4n^2 + 2$ additions.

Two divisions and one addition are required in $DECRYPT$. Thus, it is easy to see that the computational complexity of the $DECRYPT$ function is $O(1)$.

In conclusion, the client needs to undertake a total of $12n^2 + 12n$ multiplications and $4n^2 + 10n + 7$ additions. Therefore, the computational complexity of the client side of the proposed algorithm is $O(n^2)$. □

In Table 3, the number of multiplications required by every part of the proposed algorithm is demonstrated and compared with three existing algorithms from [4–6]. In Table 4, the numbers of additions required by the proposed algorithm are listed and compared in the same way as Table 3. Although the complexity of the $KEYGEN$ and $ENCRYPT$ is higher than the compared schemes [5,6], the complexities of $VERIFY$ and $DECRYPT$ in the compared schemes are higher than that of the proposed algorithm. According to [4–6], to improve security, the value of l is usually greater than 20 and the value of m is usually greater than 100, where l is the frequency of verification and m is the increase of the dimension after encryption. In fact, when l in the compared algorithms is set to 20, the security of those algorithms are very poor, which is equivalent to the security of the proposed algorithm running on a matrix of dimensions 4×4. When pursuing higher security, the local computational cost of the compared algorithms will be significantly higher than the algorithm in this paper. Therefore, it is easy to see that the proposed algorithm has the lowest local computational burden.

Table 3. Local multiplication burden.

Function	*KEYGEN*	*ENCRYPT*	*VERIFY*	*DECRYPT*	Total
Our algorithm	$8n$	$8n^2$	$4n^2 + 4n$	1	$12n^2 + 12n$
Lei's algorithm [5]	$n + m$	$2(n + m)^2$	ln^2	n	$(2 + l)n^2 + 2m^2 + 4mn + 2n + m$
Liu's algorithm [6]	$2n$	$2n^2$	ln^2	n	$(2 + l)n^2 + 3n$
Zhang's algorithm [4]	$2n$	$10n^2$	ln^2	$4n$	$(10 + l)n^2 + 6n$

Table 4. Local addition burden.

Function	*KEYGEN*	*ENCRYPT*	*VERIFY*	*DECRYPT*	Total
Our algorithm	$8n + 4$	$2n$	$4n^2 + 2$	1	$4n^2 + 10n + 7$
Lei's algorithm [5]	$2(n + m) + 1$	0	$2l(n + m)^2 - l(n + m)$	0	$2l(n + m)^2 - (l - 2)(m + n) + 1$
Liu's algorithm [6]	$2n + 1$	0	$2ln^2 - ln$	0	$2ln^2 - (l - 2)n + 1$
Zhang's algorithm [4]	$2n + 1$	$4n^2$	$2ln^2 - ln$	0	$(2l + 4)n^2 - (l - 2)n + 1$

Theorem 3. *The computational complexity of the server side of the proposed secure outsourcing algorithm for matrix determinant is $O(n^{2.373})$.*

Proof of Theorem 3. Only the *COMPUTE* function is performed by the server side. Four iterations of LU decomposition computations and $8n$ multiplications are required in Algorithm 3. If the server supposes the fastest LU decomposition algorithm (e.g., Williams' algorithm [39]), the computational overhead for the cloud side can be reduced to $O(n^{2.373})$, which has been proven in [5]. □

In comparison with the previous algorithms, the specific theoretical performance of the proposed algorithm is shown in Table 5. The nonoutsourcing algorithm for matrix determinant is $O(n^{2.373})$ using the fast LU decomposition method in paper [39]. Obviously, the local computation complexity of the client ($O(n^2)$) is substantially less than the computation complexity of nonoutsourcing ($O(n^{2.373})$).

Table 5. Theoretical computational complexity.

Function	*KEYGEN*	*ENCRYPT*	*VERIFY*	*DECRYPT*	*COMPUTE*
Our algorithm	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	$O(n^{2.373})$
Lei's algorithm [5]	$O(n + m)$	$O((n + m)^2)$	$O(ln^2)$	$O(n)$	$O(n^{2.373})$
Liu's algorithm [6]	$O(n)$	$O(n^2)$	$O(ln^2)$	$O(n)$	$O(n^{2.373})$
Zhang's algorithm [4]	$O(n)$	$O(n^2)$	$O(ln^2)$	$O(n)$	$O(n^{2.373})$

5.3. Security

A cloud server can be a passive or active attacker. Next, we will analyze the security of the proposed algorithm against both passive and active attacks.

5.3.1. Privacy against Passive Attacks

A passive attacker (semihonest) follows the procedure of the algorithm while exploiting the intermediate information to breach the privacy of matrix.

Theorem 4. *The proposed secure outsourcing algorithm for matrix determinant is privacy-protected.*

Proof of Theorem 4. It is easy to see that the methods of encrypting matrices $\boldsymbol{M_1}, \boldsymbol{M_2}, \boldsymbol{M_3}$, and $\boldsymbol{M_4}$ are consistent. The privacy input matrix $\boldsymbol{M}$ can be obtained by computing $\{\boldsymbol{M_1}, \boldsymbol{M_2}\}$ or $\{\boldsymbol{M_3}, \boldsymbol{M_4}\}$. We take $\{\boldsymbol{M_1}, \boldsymbol{M_2}\}$ as an example to prove that the proposed algorithm is privacy-protected. As $\boldsymbol{Y_1}$ and $\boldsymbol{Y_2}$ are visible to the attacker, to restore $\{\boldsymbol{M_1}, \boldsymbol{M_2}\}$, the attacker needs to guess $\boldsymbol{P_1}$, $\boldsymbol{Q_1}$, $\boldsymbol{P_2}$, and $\boldsymbol{Q_2}$. Then, the attacker uses the inverse matrices of $\boldsymbol{P_1}$, $\boldsymbol{Q_1}$, $\boldsymbol{P_2}$, and $\boldsymbol{Q_2}$ to restore $\{\boldsymbol{M_1}, \boldsymbol{M_2}\}$. When generating the original diagonal matrices $\boldsymbol{P_1}$, $\boldsymbol{Q_1}$, $\boldsymbol{P_2}$, and $\boldsymbol{Q_2}$ in the $KEYGEN$ function, a total of $4n$ elements are selected from the key space $\mathcal{K} = \{1, 0\}^{\lambda}$. The probability of attacker A correctly guessing the $4n$ elements is $\frac{1}{(2^{\lambda})^{4n}}$. Besides, from the perspective of the attacker, as long as the frequencies of mix-rows/mix-columns (n_1, n_2, m_1, m_2) are large enough, it is equivalent to repositioning the n nonzero elements of the diagonal matrix ($\boldsymbol{P_1}$, $\boldsymbol{Q_1}$, $\boldsymbol{P_2}$, $\boldsymbol{Q_2}$) to ensure that all rows/columns of the new matrix have only one element. Thus, through the mix-row/mix-column operations in lines 7–14 of Algorithm 1, the attacker successfully guessing any matrix requires $n!$ attempts. Therefore, the passive attacker should make $(n!)^4(2^{\lambda})^{4n}$ brute-force guesses to obtain $\{\boldsymbol{M_1}, \boldsymbol{M_2}\}$. Then, the attacker can easily compute $\boldsymbol{M}$. The probability that the attacker A obtains the secret input $\boldsymbol{M}$ is shown at Equation (14). When either the size of the matrix or the key space $\mathcal{K}$ is large enough, the value of $Prob_A^{\boldsymbol{M}}$ will be so small that it can be ignored.

As for the privacy of the output $det(\boldsymbol{M})$, because $det(\boldsymbol{M}) = \frac{r_1}{t_1} + \frac{r_2}{t_2}$, the attacker must obtain t_1, t_2 before computing $det(\boldsymbol{M})$. As $t_1 = det(\boldsymbol{P_1})det(\boldsymbol{Q_1}), t_2 = det(\boldsymbol{P_2})det(\boldsymbol{Q_2})$, computing t_1, t_2 is equivalent to guessing $\boldsymbol{P_1}$, $\boldsymbol{Q_1}$, $\boldsymbol{P_2}$, and $\boldsymbol{Q_2}$. Thus, the probability that the attacker A obtains the secret output $det(\boldsymbol{M})$ is the same as obtaining the input privacy.

Thus, we can conclude that the proposed secure outsourcing algorithm for matrix determinant is privacy-protected.

$$Prob_A^{\boldsymbol{M}}(\boldsymbol{Y_1}, \boldsymbol{Y_2}) = \frac{1}{(n!)^4(2^{\lambda})^{4n}} \rightarrow negli \tag{14}$$

□

5.3.2. Security against Active Attacks

An active attacker (malicious) injects false computation results into the algorithm to tamper with the whole procedure.

Theorem 5. *The proposed secure outsourcing algorithm for matrix determinant is cheating-detected.*

Proof of Theorem 5. There are three types of attacks with different complexity levels.

In the first attack, the attacker returns random $r_1', r_2', r_3', r_4', \boldsymbol{L_1'}, \boldsymbol{L_2'}, \boldsymbol{U_1'}, \boldsymbol{U_2'}$ to the client with $O(1)$ computational complexity. Obviously, $r_1' \neq \prod_{i=1}^{n} L_1'(i,i)U_1'(i,i)$, $r_3' \neq \prod_{i=1}^{n} L_3'(i,i)U_3'(i,i)$, and $\frac{r_1'}{t_1} + \frac{r_2'}{t_2} \neq \frac{r_3'}{t_3} + \frac{r_4'}{t_4}$. Thus, it cannot pass the verifications of Equations (5)–(7). The malicious cloud can also perform a small number of computations with $O(n)$ computational complexity so that $r_1' = \prod_{i=1}^{n} L_1'(i,i)U_1'(i,i)$ and $r_3' = \prod_{i=1}^{n} L_3'(i,i)U_3'(i,i)$, which can nullify the verifications of Equations (5) and (6). However, due to the lack of t_1, t_2, t_3, t_4, it still fails to pass the verification of Equation (7).

The complexity of the second type of attack is $O(n^{2.373})$. There are two ways of attacking. In the first way, the attacker computes the correct results σ_y, but chooses a random ath element on the diagonal of $\boldsymbol{L_1}$ or $\boldsymbol{U_1}$ and tampers with it (e.g., $L_1'(a,a) = \gamma L_1(a,a)$). In the same way, the attacker tampers with a random bth element on the diagonal of $\boldsymbol{L_3}$ or $\boldsymbol{U_3}$ (e.g., $L_3'(b,b) = \gamma L_3(b,b)$). Besides, the attacker also changes the r_1, r_2, r_3, r_4 by $r_1' = \gamma r_1, \ldots, r_4' = \gamma r_4$. The above attack returns $r_1', r_2', r_3', r_4', L_1', U_1, L_3', U_3$ to the client, which can successfully nullify the verifications of Equations (5)–(7). However, it cannot pass the verification in lines 6–17 of the $VERIFY$ function. The verifications in lines 6–17 verify all the diagonal elements in $\boldsymbol{L_1}$, $\boldsymbol{L_3}$, $\boldsymbol{U_1}$, and $\boldsymbol{U_3}$ at least once. The nature of the verifications in lines 6–17 is to select $2n$ elements in the matrices $\boldsymbol{Y_1}$ and $\boldsymbol{Y_3}$, respectively, to verify the correctness of the diagonal elements in $\boldsymbol{L_1}$, $\boldsymbol{L_3}$, $\boldsymbol{U_1}$, and $\boldsymbol{U_3}$. As shown in Equations (15) and (16), the error in the ath/bth term in $\boldsymbol{l_1'}(a,-)/\boldsymbol{l_3'}(b,-)$ will be propagated to $Y_1'(a,j)/Y_3'(b,k)$, which is not equal to $Y_1(a,j)/Y_3(b,k)$. Thus, the forged $L_1'(a,a)$ and $L_3'(a,a)$ can be certainly detected.

$$Y_1(a,j) = \boldsymbol{l_1(a,-)\cdot u_1(-,j) \neq l_1'(a,-)\cdot u_1(-,j)}, (a \leq j \leq n) \tag{15}$$

$$Y_3(b,k) = \boldsymbol{l_3(b,-)\cdot u_3(-,k) \neq l_3'(b,-)\cdot u_1(-,k)}, (b \leq k \leq n) \tag{16}$$

In the second way, before performing the *COMPUTE* function, the attacker tampers with a random element in $\boldsymbol{Y_1}$ and $\boldsymbol{Y_3}$, respectively (Tampering with more items will be easier to detect.). Then, the attacker performs the *COMPUTE* function with the forged input $\boldsymbol{Y_1', Y_3'}$, and returns the cheating result $\sigma_y' = r_1', r_2, r_3', r_4, L_1', U_1', L_3', U_3'$ to the client. As the verification in lines 6–17 can only detect this attack with a probability of $\frac{2}{n}$. This way of attacking can easily nullify the verification in lines 6–17. However, as shown in Equation (17), it cannot pass the verification of Equation (7). Forging more elements in $\boldsymbol{Y_1, Y_3}$, or elements in $\boldsymbol{Y_2, Y_4}$ can also be detected by Equation (7) (e.g., the cloud returns $\sigma_y' = r_1, r_2', r_3', r_4', L_1, U_1, L_3', U_3'$ to the client where r_i' is the cheating result and r_i is the correct result).

$$\frac{r_1'}{t_1} + \frac{r_2}{t_2} \neq \frac{r_3'}{t_3} + \frac{r_4}{t_4} \tag{17}$$

The complexity of the third attack is much greater than others. In order to pass all the verification, before attacking, the attacker has to make $(n!)^4$ brute-force guesses to get the $4n$ positions that the client verifies in $\boldsymbol{Y_1}$ and $\boldsymbol{Y_3}$ (lines 6–17). Then, the cloud constructs two forged matrices $\boldsymbol{Y_1'}$ and $\boldsymbol{Y_3'}$. The forged matrices $\boldsymbol{Y_1'}$ and $\boldsymbol{Y_3'}$ satisfy the condition that the elements at the detected positions remain unchanged, and the elements at other positions are changed. Meanwhile, the attacker ensures $det(\boldsymbol{Y_1'}) = \gamma det(\boldsymbol{Y_1})$, $det(\boldsymbol{Y_3'}) = \gamma det(\boldsymbol{Y_3})$. Afterwards, the cloud tampers with r_1, r_2, r_3, r_4 by $r_1' = \gamma r_1, \ldots, r_4' = \gamma r_4$. The third attack is the only way to pass all the verifications in *VERIFY* with a probability of $\frac{1}{(n!)^4}$ and return a forged result $y' = \gamma y$. The probability that the client C successfully detects the forged result is shown in Equation (18), which infinitely approaches 1 when n is large enough. Thus, we can conclude that the proposed secure outsourcing algorithm for matrix determinant is cheating-detected, when the size of the matrix is large enough.

$$Prob_C^{forge}(r_1, r_2, r_3, r_4, \boldsymbol{L_1, L_3, U_1, U_3}) = (1 - \frac{1}{(n!)^4}) \to 1 \tag{18}$$

□

As shown in Table 6, the theoretical security of the proposed algorithm is significantly higher than the other three algorithms while using the same key space $\mathcal{K} = \{1,0\}^\lambda$. According to [4–6], the parameter l is recommended to be greater than 20. Thus, when $n \geq 5$, the proposed algorithm can achieve a high cheating detectability comparable to these three algorithms with the lowest computational cost, as demonstrated in Tables 3 and 4. Actually, it is common to compute the determinant of the large matrices with dimensions of far-greater than 5 in DIP and machine learning.

Table 6. Probability of successful attack.

Attack	Our Algorithm	Liu's Algorithm [6]	Lei's Algorithm [5]	Zhang's Algorithm [4]
Privacy Leakage	$\frac{1}{(n!)^4(2^\lambda)^{4n}}$	$\frac{1}{(n!)^2(2^\lambda)^{2n}}$	$\frac{1}{((n+m)!)^2(2^\lambda)^{2(n+m)}}$	$\frac{1}{(n!)^2(2^\lambda)^{10n-8}}$
Result cheating	$\frac{1}{(n!)^4}$	$\frac{1}{(2)^l}$	$\frac{1}{(2)^l}$	$\frac{1}{(2)^l}$

6. Performance Evaluation

According to the above theoretical analysis, our algorithm can significantly reduce the local computational burden of the client. In this section, we implement the algorithm to assess its practical efficiency. The client and the cloud server functions in our experiments were conducted on the same machine, which has an Intel(R) Core(TM) i7-8550U CPU 1.80 GHz with eight cores. We implemented the proposed algorithm by Matlab and used the LAPACK [40] package to perform the LU decomposition. The communication costs between the client and the server were ignored, since the computations dominate the running time. Table 7 shows the notations used in this section.

Table 7. Notations in experiments.

Notations	Implication
t_o	The time consumption of nonoutsourcing scheme.
t_s	The time consumption of cloud.
t_{c1}	The time consumption of client in key generation and encryption.
t_{c2}	The time consumption of client in decryption and verification.
t_c	The time consumption of client.
$\frac{t_o}{t_c}$	Acceleration ratio of client.

Our goal is to reduce the client's computational burden through outsourcing. Therefore, the ratio of time consumption without outsourcing to time consumption with outsourcing is an important measure, which is referred to as the *Acceleration Ratio* of clients.

In our comparisons, we only considered the existing algorithms proposed for the malicious model, without considering those merely for the semihonest model, since the former model is more secure. As far as we know, the currently existing algorithms for the malicious model include only Lei's algorithm [5], Liu's algorithm [6], and Zhang's algorithm [4]. In the experiment, we set the parameter l to 20 in the three previous algorithms, and set the parameter m to 500 in Lei's algorithm. We compared the running time of every part of the algorithms on matrices of different dimensions. Table 8 and Figure 3 show that our algorithm has the highest acceleration rate on matrices of all dimensions. The time consumption on cloud in our algorithm is slightly higher than the three previous algorithms, which is not a big issue since this follows the aim of outsourcing by moving computational burden from local to cloud. As shown in Figure 4, compared with the previous algorithms and nonoutsourcing algorithms, our proposed algorithm has the lowest local computational burden. As for the security level, even if l is set to 20, the security of the three previous algorithms are still much lower than the proposed algorithm.

In order to prove that the proposed algorithm achieves a higher security level with less local computation costs. we also compare the local running time of the proposed algorithm with the previous three algorithm on different values of parameter l. In this experiment, the dimension of matrix is fixed at 2000. As shown in Figure 5, because the proposed algorithm uses a new verification method, which does not involve the parameter l, the local running time of the proposed algorithm remains unchanged. As the parameter l increases, the cheating detectability of the previous three algorithms increases, since the forged results can escape local verifications with the probability of $\frac{1}{2^l}$, while in our algorithm the cheating detectability remains at the probability of $1 - \frac{1}{(2000!)^4}$. However, the local computational burden of the three algorithms also increases significantly.

We also apply the proposed secure outsourcing algorithm as a basic module to the Cramer's rule to solve linear equations. We compare the time consumption of outsourcing determinant computation and nonoutsourcing in solving linear equations of different dimensions. As shown in Figure 6, outsourcing the computations of determinant can significantly reduce the time consumption of solving linear equations, which also proves the efficiency superiority of the proposed secure outsourcing algorithm for matrix determinant computation.

Table 8. Experimental results (time in milliseconds).

Algorithm	Dimension	t_o	t_{c1}	t_{c2}	t_c	t_s	$\frac{t_o}{t_c}$
Our algorithm	500	32.829	14.188	3.547	17.735	49.264	1.851
	2000	290.013	94.517	31.670	126.187	352.725	2.298
	3000	606.768	191.122	41.954	233.078	903.288	2.603
	5000	2410.926	470.294	148.515	618.809	2982.729	3.896
	8000	4666.675	808.776	228.117	1036.893	6163.738	4.501
	10,000	12,818.648	1960.510	459.867	2420.337	15,895.125	5.296
Lei's algorithm [5]	500	32.829	1.747	25.184	26.931	45.775	1.219
	2000	290.013	5.528	186.026	191.554	348.904	1.514
	3000	606.768	10.951	341.002	351.953	894.031	1.724
	5000	2410.926	42.309	974.528	1016.837	2767.491	2.371
	8000	4666.675	274.662	1198.869	1473.532	5495.207	3.167
	10,000	12,818.648	796.518	2779.116	3575.634	14,175.942	3.585
Liu's algorithm [6]	500	32.829	0.793	19.764	20.557	38.379	1.597
	2000	290.013	4.923	152.771	157.694	311.684	1.839
	3000	606.768	7.595	258.414	266.009	894.031	2.281
	5000	2410.926	30.519	669.924	701.443	2531.673	3.442
	8000	4666.675	132.731	985.303	1118.034	4951.749	4.174
	10,000	12,818.648	551.634	2024.461	2576.095	13,667.593	4.976
Zhang's algorithm [4]	500	32.829	18.475	20.498	38.973	37.517	0.842
	2000	290.013	104.772	167.538	272.310	324.941	1.065
	3000	606.768	207.462	277.737	485.199	875.963	1.251
	5000	2410.926	522.743	720.335	1243.078	2539.492	1.939
	8000	4666.675	983.769	1142.779	2126.548	5027.820	2.194
	10,000	12,818.648	2647.539	3176.957	5824.496	12,741.742	2.201

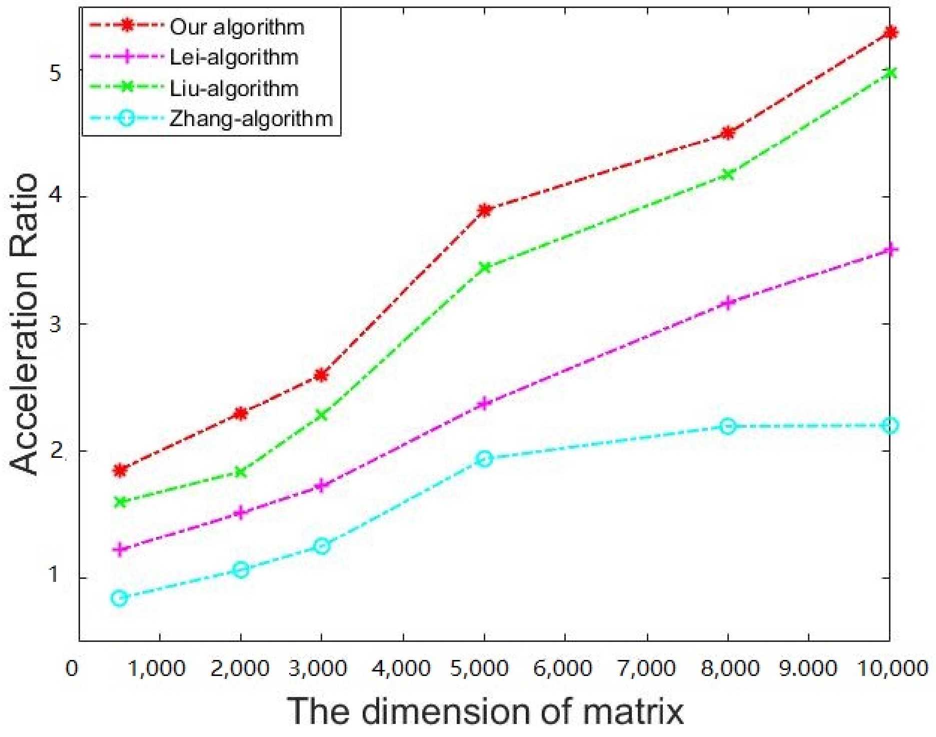

Figure 3. Acceleration ratio of the client.

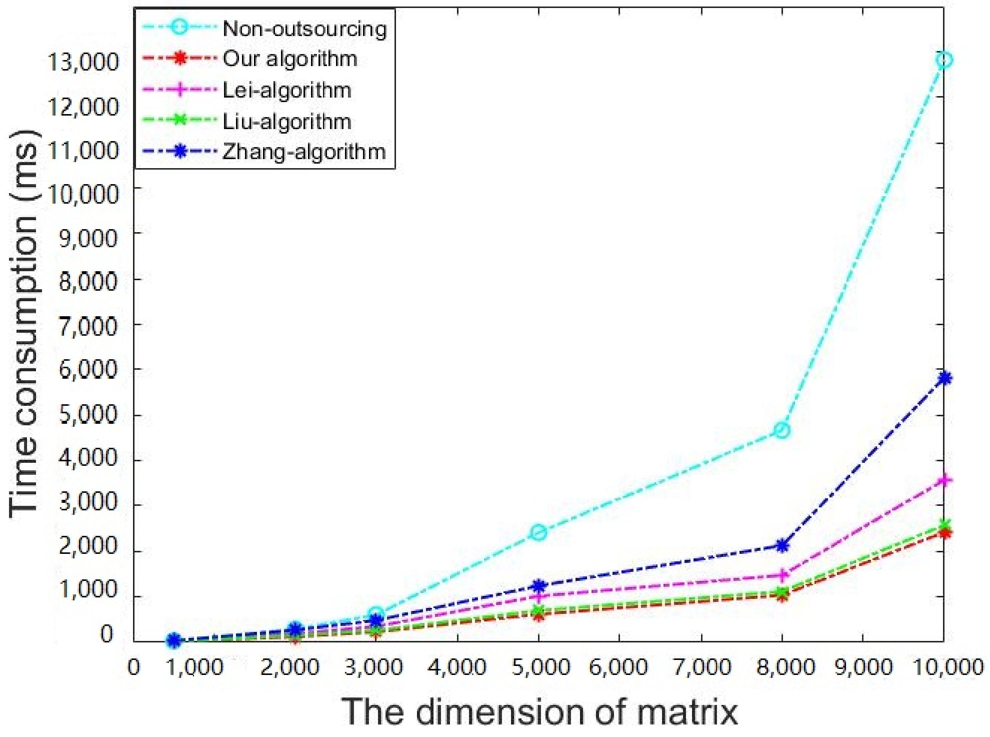

Figure 4. The time consumption of the client.

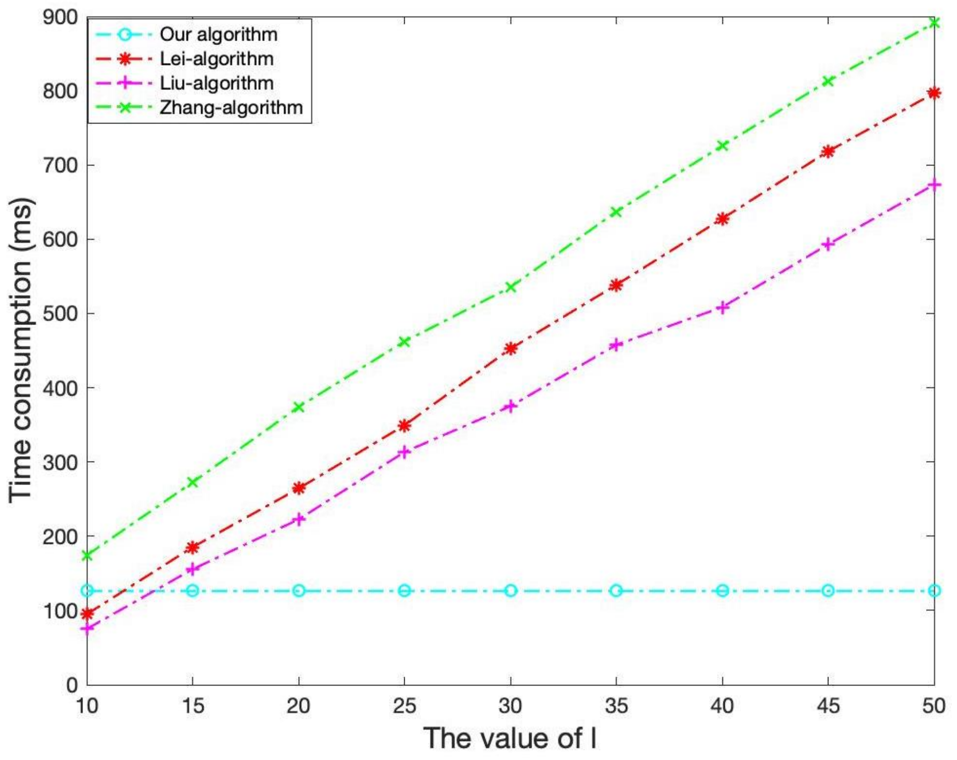

Figure 5. The time consumption of the client when using different parameter l.

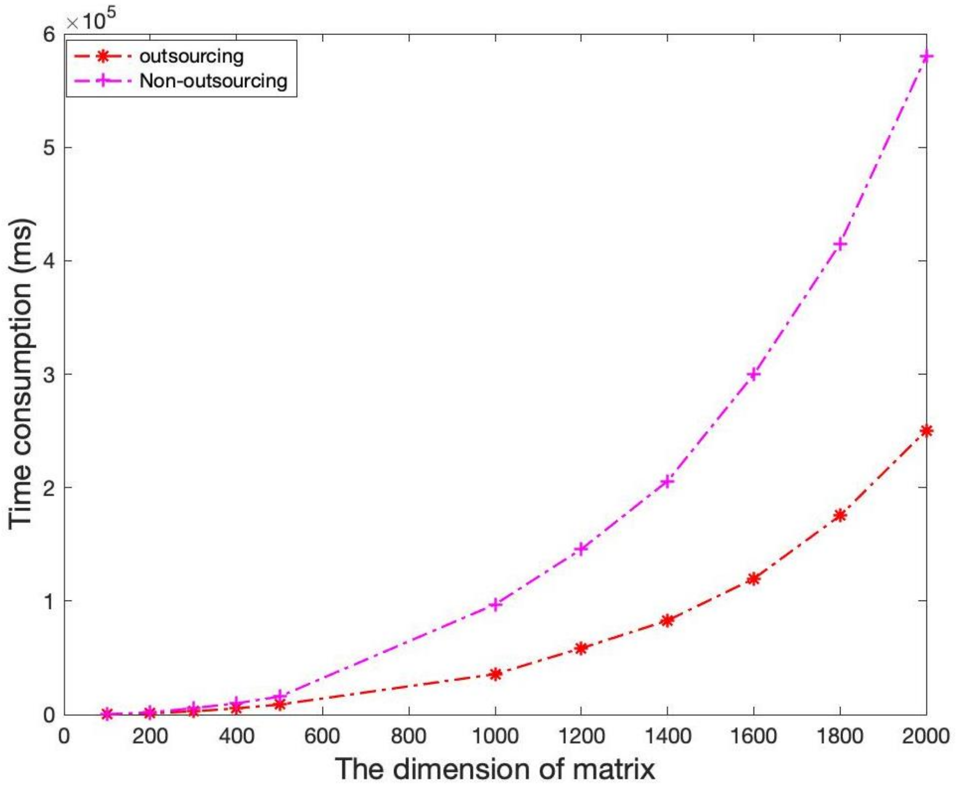

Figure 6. The time consumption of solving linear equations.

7. Conclusions

In this paper, we propose a new secure outsourcing algorithm for matrix determinant computation under the malicious model. We also conduct theoretical analysis to prove the correctness, efficiency, privacy protection level, and cheating detectability for the proposed algorithm. In comparison with the previous algorithms in [4–6], the proposed algorithm achieves higher cheating detectability with less computations on the client. The previous algorithms [4–6] use Freivald's method [38] for verification, which achieves a high cheating detectability by continuously increasing local computational burden. The cheating detectability of the proposed algorithm does not depend on the frequency of verification but is only related to the size of the matrix. Even in the case that the dimension of the matrix is not large enough (greater than or equal to 5), the cheating detectability of our algorithm is significantly better than the previous algorithms. For the privacy, the state-of-the-art algorithm [4] has the highest privacy, but its local computation cost usually nullifies the efficiency benefit of outsourcing when the dimension of matrix is less than 2000. Our algorithm with the lowest local computation cost achieves privacy comparable with the state-of-the-art works. We also conduct experiments to demonstrate the local efficiency superiority of the proposed algorithm. However, the theoretical analysis and experimental results also show that the proposed algorithm has a higher cloud computation cost than the three previous algorithms, which is not a big issue since this follows the aim of outsourcing by moving computational burden from local to cloud. As future work, we will study how to reduce the cloud's computational burden in the secure outsourcing algorithms for computations of the matrix.

Author Contributions: Conceptualization, M.S. and Y.S.; methodology, M.S.; software, M.S.; validation, M.S. and Y.S.; formal analysis, M.S. and Y.S.; investigation, M.S.; resources, M.S. and Y.S.; data curation, M.S.; writing—original draft preparation, M.S.; writing—review and editing, Y.S.; visualization, M.S.; supervision, Y.S.; project administration, Y.S.; funding acquisition, Y.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Science and Technology Program of Guangzhou, China (Grant No. 201904010209), and the Science and Technology Program of Guangdong Province, China (Grant No. 2017A010101039).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Brunette, G.; Mogull, R. Security guidance for critical areas of focus in cloud computing v2.1 *Cloud Secur. Alliance* **2017**, 1–76. Avaliable online: http://www.cloudsecurityalliance.org/csaguide.pdf (accessed on 4 September 2021).
2. Bolton, T.; Dargahi, T.; Belguith, S.; Al-Rakhami, M.S.; Sodhro, A.H. On the security and privacy challenges of virtual assistants. *Sensors* **2021**, *21*, 2312. [CrossRef] [PubMed]
3. Goldreich, O.; Micali, S.; Wigderson, A. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*; ACM: New York, NY, USA, 2019; pp. 307–328.
4. Zhang, S.; Tian, C.; Zhang, H.; Yu, J.; Li, F. Practical and Secure Outsourcing Algorithms of Matrix Operations Based on a Novel Matrix Encryption Method. *IEEE Access* **2019**, *7*, 53823–53838. [CrossRef]
5. Lei, X.; Liao, X.; Huang, T.; Li, H. Cloud computing service: The caseof large matrix determinant computation. *IEEE Trans. Serv. Comput.* **2014**, *8*, 688–700. [CrossRef]
6. Liu, J.; Bi, J.; Li, M. Secure outsourcing of large matrix determinant computation. *Front. Comput. Sci.* **2020**, *14*, 1–12. [CrossRef]
7. Nykvist, C.; Larsson, M.; Sodhro, A.H.; Gurtov, A. A lightweight portable intrusion detection communication system for auditing applications. *Int. J. Commun. Syst.* **2020**, *33*, e4327. [CrossRef]
8. Kayes, A.; Kalaria, R.; Sarker, I.H.; Islam, M.; Watters, P.A.; Ng, A.; Hammoudeh, M.; Badsha, S.; Kumara, I.; et al. A survey of context-aware access control mechanisms for cloud and fog networks: Taxonomy and open research issues. *Sensors* **2020**, *20*, 2464. [CrossRef]
9. Brakerski, Z. Fully homomorphic encryptionwithout modulus switching from classical GapSVP. In *Annual Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 868–886.

10. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. TFHE: Fast fully homomorphic encryption over the torus. *J. Cryptol.* **2020**, *33*, 34–91. [CrossRef]
11. Shen, T.; Wang, F.; Chen, K.; Wang, K.; Li, B. Efficient leveled (multi) identity-based fully homomorphic encryption schemes. *IEEE Access* **2019**, *7*, 79299–79310. [CrossRef]
12. Da Silva, D.W.; de Araujo, C.P.; Chow, E.; Barillas, B.S. A new approach towards fully homomorphic encryption over geometric algebra. In Proceedings of the 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 10–12 October 2019; pp. 0241–0249.
13. Li, J.; Yu, Q.; Zhang, Y.; Shen, J. Key-policy attribute-based encryption against continual auxiliary input leakage. *Inf. Sci.* **2019**, *470*, 175–188. [CrossRef]
14. Waters, B. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *International Workshop on Public Key Cryptography*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 53–70.
15. Attrapadung, N. Unbounded dynamic predicate compositions in attribute-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 34–67.
16. Fu, A.; Li, S.; Yu, S.; Zhang, Y.; Sun, Y. Privacy-preserving composite modular exponentiation outsourcing with optimal checkability in single untrusted cloud server. *J. Netw. Comput. Appl.* **2018**, *118*, 102–112. [CrossRef]
17. Su, Q.; Zhang, R.; Xue, R. Secure outsourcing algorithms for composite modular exponentiation based on single untrusted cloud. *Comput. J.* **2020**, *63*, 1271–1271. [CrossRef]
18. Zhou, Q.; Tian, C.; Zhang, H.; Yu, J.; Li, F. How to securely outsource the extended euclidean algorithm for large-scale polynomials over finite fields. *Inf. Sci.* **2020**, *512*, 641–660. [CrossRef]
19. Ren, Y.; Ding, N.; Wang, T.; Lu, H.; Gu, D. New algorithms for verifiable outsourcing of bilinear pairings. *Sci. China Inf. Sci.* **2016**, *59*, 1–3. [CrossRef]
20. Lin, C.; He, D.; Huang, X.; Xie, X.; Choo, K.K.R. Blockchain-based system for secure outsourcing of bilinear pairings. *Inf. Sci.* **2020**, *527*, 590–601. [CrossRef]
21. Tong, L.; Yu, J.; Zhang, H. Secure Outsourcing Algorithm for Bilinear Pairings without Pre-Computation. In Proceedings of the 2019 IEEE Conference on Dependable and Secure Computing (DSC), Hangzhou, China, 18–20 November 2019; pp. 1–7.
22. Song, M.; Sang, Y.; Zeng, Y.; Luo, S. Blockchain-Based Secure Outsourcing of Polynomial Multiplication and Its Application in Fully Homomorphic Encryption. *Secur. Commun. Netw.* **2021**, *2021*, 9962575.
23. Zhang, Y.; Blanton, M. Efficient secure and verifiable outsourcing of matrix multiplications. In *International Conference on Information Security*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 158–178.
24. Kumar, M.; Mishra, V.; Shukla, A.; Singh, M.; Vardhan, M. A novel publicly delegable secure outsourcing algorithm for large-scale matrix multiplication. *J. Intell. Fuzzy Syst.* **2020**, *38*, 6445–6455. [CrossRef]
25. Wang, S.; Huang, H. Secure outsourced computation of multiple matrix multiplication based on fully homomorphic encryption. *KSII Trans. Internet Inf. Syst. (TIIS)* **2019**, *13*, 5616–5630.
26. Wu, Y.; Liao, Y.; Liang, Y.; Liu, Y. Secure and Efficient Protocol for Outsourcing Large-Scale Matrix Multiplication to the Cloud. *IEEE Access* **2020**, *8*, 227556–227565. [CrossRef]
27. Duan, J.; Zhou, J.; Li, Y. Secure and verifiable outsourcing of nonnegative matrix factorization (NMF). In Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security, Vigo, Spain, 20–22 June 2016; pp. 63–68.
28. Liu, Z.; Li, B.; Han, Q. Secure and verifiable outsourcing protocol for non-negative matrix factorisation. *Int. J. High Perform. Comput. Netw.* **2018**, *11*, 14–23. [CrossRef]
29. Fu, A.; Chen, Z.; Mu, Y.; Susilo, W.; Sun, Y.; Wu, J. Cloud-based outsourcing for enabling privacy-preserving large-scale non-negative matrix factorization. *IEEE Trans. Serv. Comput.* **2019**. [CrossRef]
30. Duan, J.; Zhou, J.; Li, Y. Secure and verifiable outsourcing of large-scale nonnegative matrix factorization (NMF). *IEEE Trans. Serv. Comput.* **2019**. [CrossRef]
31. Hu, C.; Alhothaily, A.; Alrawais, A.; Cheng, X.; Sturtivant, C.; Liu, H. A secure and verifiable outsourcing scheme for matrix inverse computation. In Proceedings of the IEEE INFOCOM 2017-IEEE Conference on Computer Communications, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
32. Pan, S.; Wang, Q.; Zheng, F.; Dong, J. Secure and efficient outsourcing of large-scale matrix inverse computation. In *International Conference on Wireless Algorithms, Systems, and Applications*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 374–386.
33. Chen, Z.; Fu, A.; Xiao, K.; Su, M.; Yu, Y.; Wang, Y. Secure and verifiable outsourcing of large-scale matrix inversion without precondition in cloud computing. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.
34. Pramkaew, C.; Ngamsuriyaroj, S. Lightweight scheme of secure outsourcing SVD of a large matrix on cloud. *J. Inf. Secur. Appl.* **2018**, *41*, 92–102. [CrossRef]
35. Chen, J.; Liu, L.; Chen, R.; Peng, W. SHOSVD: Secure Outsourcing of High-Order Singular Value Decomposition. In *Australasian Conference on Information Security and Privacy*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 309–329.
36. Kim, D.; Son, Y.; Kim, D.; Kim, A.; Hong, S.; Cheon, J.H. Privacy-preserving approximate GWAS computation based on homomorphic encryption. *BMC Med. Genom.* **2020**, *13*, 77. [CrossRef] [PubMed]
37. Zong, H.; Huang, H.; Wang, S. Secure Outsourced Computation of Matrix Determinant Based on Fully Homomorphic Encryption. *IEEE Access* **2021**, *9*, 22651–22661. [CrossRef]

38. Freivalds, R. Probabilistic Machines Can Use Less Running Time. In Proceedings of the IFIP Congress, Toronto, ON, Canada, 8–12 August 1977; Volume 839, p. 842.
39. Chen, Y.; Nguyen, P.Q. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 502–519.
40. Anderson, E.; Bai, Z.; Bischof, C.; Blackford, L.S.; Demmel, J.; Dongarra, J.; Du Croz, J.; Greenbaum, A.; Hammarling, S.; McKenney, A.; et al. *LAPACK Users' Guide*; SIAM: Philadelphia, PA, USA, 1999.