Verifiable Outsourcing Computation for Matrix Multiplication With Improved Efficiency and Applicability

Shenmin Zhang, *Student Member, IEEE*, Hongwei Li, *Member, IEEE*, Yuanshun Dai, *Senior Member, IEEE*, Jin Li, *Member, IEEE*, Miao He, *Member, IEEE*, and Rongxing Lu, *Senior Member, IEEE*

***Abstract*—In recent years, the rapid development of Internet of Things (IoT) and big data shows the huge demand of outsourcing computing in cloud computing to assist some clients with low ability devices to fulfill massive data processing. In the mean time, considering the uncertainty of the Internet devices, we should persist security and high-efficiency. Uploaded data and returned results should be protected from attack of adversary, which guarantees the security, while efficiency requires low overhead of clients to finish the whole calculation procedure. In this paper, we main focus on the public verifiable outsourcing scheme on matrix multiplication, which can be applied in many IoT scenes, such as path planning and aggregation operation in Internet of Vehicles and smart grids, respectively. Specially, this paper presents schemes for two different functions of matrix multiplication, which strengthens the applicability. Moreover, security analysis and performance evaluation in this paper properly present the superiority of this paper.**

***Index Terms*—Applicability, efficiency, performance, Internet of Things (IoT), matrix multiplication, outsourcing scheme, security.**

I. Introduction

As THE development of Internet of Things (IoT) [1], the devices with weak ability in computation are not limited in traditional devices. Some devices. Example includes smart thermostats, LED light bulbs, programmable and intelligent cars [2]–[4], which may perform computations with the help of cloud computing [5]–[8] to achieve specific tasks, such as drone navigation, intelligent transportation and so on. Accordingly, outsourcing computing utilizing cloud, has been increasingly approved to execute users' complex task due to advantage of tremendous compute power, and further promoted the rapid development of IoT, where the client devices can acquire the on-demand service for computation from the cloud server with huge computation ability.

Naturally, as the cloud server is a untrusted platform [9]–[12], security of data in IoT is becoming a major problem. Besides, data generated by the IoT, verification of results correctness becomes increasingly important. While uploaded data, i.e., inputs and outputs, may contain sensitive information, the client must be convinced of the security of those data [13]–[15]. To assure the correctness of results returned by the untrusted server, the results will be verified utilizing verification proofs. Crucially, for the client, the process of verification and data protection must be quite compared with the process of function operation.

In delegation, most work of verification is conducted by the client. However, it is necessary to verify the results by any other trusted third party [16]. For instance, the computation results should be verified by several clients with different secret keys, or the results are demanded by the other party. In IoT, the third party can be some regulators. Public verification prove to be useful in applications and simultaneously alleviates the computation cost of the client.

In our scheme, we consider the outsourcing computation of matrix multiplication, which can be applied for many aspects [17]–[20], e.g., image processing [21], large-dataset processing [22], the brain model in artificial intelligence. Considering some scenes of IoT, algorithms of aggregation operation in smart grid and path planning in pilotless devices also need outsourcing computation of matrix multiplication. As the increasing demand of matrix multiplication, the relevant delegation researches are meaningful.

A. Related Work

With the fast development of outsourcing computation [23], [24], several schemes focus on this research orientation. Hohenberger and Lysyanskaya [25] proposed an outsourcing scheme for modular exponentiation. Although

Manuscript received February 21, 2018; revised April 30, 2018 and June 11, 2018; accepted August 12, 2018. Date of publication August 29, 2018; date of current version January 16, 2019. This work was supported by the National Key R&D Program of China under Grant 2017YFB0802300 and Grant 2017YFB0802000, the National Natural Science Foundation of China under Grant 61802051, Grant 61772121, Grant 61728102, and Grant 61472065, and by the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2015J056. *(Corresponding author: Hongwei Li.)*

S. Zhang is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China, and also with the CETC Big Data Research Institute Company, Ltd., Guiyang 550022, China.

H. Li is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China, and also with the Science and Technology on Communication Security Laboratory, Chengdu 610041, China (e-mail: hongwei.uestc@gmail.com).

Y. Dai is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China

J. Li is with the School of Computer Science, Guangzhou University, Guangzhou 510006, China.

M. He is with Fortinet Technologies (Canada) ULC, Ottawa, ON K2H 8G3, Canada.

R. Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada.

Digital Object Identifier 10.1109/JIOT.2018.2867113

in this paper, cloud computing does not appear extensively and is low in efficiency, this scheme still enlightens the following work. Atallah and Frikken [26] delegated expensive linear algebraic computations, especially for large matrices, with security and privacy. In the same year, combining fully homomorphic encryption (FHE) [27] and Yao' garbled circuit [28], Gennaro *et al.* [10] proposed a scheme. This scheme provides the formalized definition of noninteractive verifiable outsourcing computation using the pseudo-random functions (PRF). Although FHE is widely applied in data encryption for delegation schemes [10], [29]–[36], the computation overhead of this approach is too huge for application. Backes *et al.* [30] proposed a novel cryptographic scheme for quadratic polynomials, a class of computations over a large number of variables. However, it is hard for this scheme to be applied for some special functions [37]. Wang *et al.* [38] proposed a secure outsourcing scheme of widely applicable linear programming computations, which is greatly helpful in the future.

In our scheme, we focus on the delegation of matrix multiplication as many other schemes do. In [29], Fiore's scheme provides a public verifiable delegation for linear operation, e.g., matrix multiplication. This scheme does not give the concrete encryption scheme of FHE, while the high-efficiency cannot be promised using FHE and PRF. Schemes [39] and [40] both improve [29] in matrix multiplication with a constant matrix. Feng and Safavi-Naini [39] showed an encryption scheme, while it also brings problems in computation overhead. Fiore and Gennaro [29] optimized the process in producing verification tags. In this scheme, the efficiency can be improved to some extent, while the encryption scheme has some drawbacks. Furthermore, aiming at matrix multiplication with two alterable matrices, Jia *et al.* [41] encrypted matrices ingeniously. But the outputs of its algorithm are exposed to the cloud server entirely, thus data privacy cannot be guaranteed. Nowadays, many researches are continually devoting their efforts to available applications.

B. Our Contribution

In this paper, we focus on the outsourcing computation of matrix multiplication with the consideration of both efficiency and security for the whole scheme. In our previous research, we presents a delegation scheme (EPP-DMM) for one-matrix-constant matrix multiplication in amortized model. Though under the premise of security and efficiency, the security analysis of original paper is not adequate and the efficiency also has room for improvement.

As a development, in this paper, the main contributions can be described as follows.

1) The applicability of this paper is expanded, while this paper can be applied for two different functions of matrix multiplication. As the outsourcing function of original paper is matrix multiplication with one constant matrix, this paper successfully expands the scheme for another matrix multiplication, of which two matrices are variable.
2) The efficiency of this paper is further enhanced compared with original version and other schemes. Changing the verification process, the scheme can be improved especially for computing overhead.
3) In security, inputs and outputs can be protected both for the two different functions. Meanwhile, we present the precise security analysis for the unforgeability of proof.

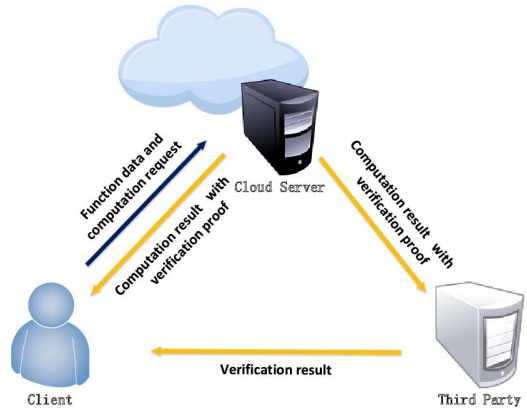


Fig. 1. System model.

The rest of this paper is structured as follows. To present the basic condition our algorithm faced, we describe the problem formulation in Section II, which contains security requirements, system, and function models. In Section III, we shows definitions for our scheme, while Section IV presents the detail of our scheme. Besides, this paper conducts security analysis and performance evaluation in Sections V and VI, respectively, to support itself. In the end of Section VII, we conclude this paper.

II. Problem Formulation

A. System Model

As shown in Fig. 1, the system model, in this paper, has three entities.

1) *Client:* As a party with limited computing resources, the client can outsource the heavy task to the other parties, i.e., the cloud server. Therefore, before sending the computation request to the cloud server, the client will upload the computation data of requesting function to the server in advance, and then generates the confidential key SK and the published key PK. Utilizing the keys, the client can generate the verification data VT and V to prove the result, and conduct encryption and decryption mechanism to insure the privacy of the inputs and outputs. If necessary, the client will verify the result by itself through the data returned by the server.
2) *Cloud Server:* With the considerable ability in storage and computation, the server can not only can storage huge data for the client, but also conduct calculation for it. When receiving the request from the client, the server computes the function results together with the corresponding verification proof VP, both of which will be sent to the client or the third party.
3) *Third Party:* The third party, which is credible, can conduct verification produce for the client when necessary.

Utilizing the verification data, it can verify the result provided by the cloud server.

B. Function Model

In this paper, we design two schemes for two different functions of matrix multiplication. In this paper, both two functions are conducted in amortized model, which means that we focus on computing for many times. As it is difficult for client to compute the operations of huge data, it is necessary to outsource the matrix multiplication to the server. According to the amortized model in which the client can conduct a one-time expensive phase in advance and proceed with multiple times of online phase as request, those two matrix multiplication functions can be described as follows.

1) Matrix Multiplication With Constant Matrix: We define a function $Y = F(X) = MX$, where M is fixed in matrix multiplication and the input X changes according to the demand of the client. In the amortized model of delegation, this function can be requested for many times. Due to the fixed matrix M, in the preprocessing phase, some data can be calculated only for one time in advance. When the client needs the function result of input X, it only generates some data related to the input X.

2) Matrix Multiplication With Two Alterable Matrices: Here, we present another function $Y = F(X^{(1)}, X^{(2)}) = X^{(1)} \cdot X^{(2)}$, in which the matrices of multiplication is alterable obviously. Hence, under the amortized function model, the inputs $X^{(1)}$ and $X^{(2)}$ can be chosen from the databases $D^{(1)}$ and $D^{(2)}$ [42], respectively, which means inputs of function $F(X^{(1)}, X^{(2)})$ are all from databases, where each matrix is remarked with identity id. This kinds of function can be applied to the multiplication of matrices with limited number. Considering the one-time cost of preprocessing phase, all matrices of databases $D^{(1)}$ and $D^{(2)}$ together with the corresponding verification data are disposed before being uploaded. When requesting, the client sends tuple (id_1, id_2) as the computation query of function $F(X^{(1)}, X^{(2)})$.

We define an inquiry mechanism that the information, having been requested id tuple together with corresponding computation results and verification proofs, will be kept in the cloud server. If (id_1, id_2) has been requested, the server can return the results and proofs without computation, otherwise, it computes for the client.

C. Security Requirements

As the security of outsourcing scheme is badly needed all the way, we also consider it in this paper, where the security requirements are summarized into two aspects.

1) *Data Privacy:* As inputs and outputs may contain sensitive data of the client, if them are exposed to the cloud server, there is a great risk for those data to be abused by the spiteful adversary. In case of revealing privacy of itself, the client protects the inputs through encryption before uploading, and decrypts output in the end.
2) *Proof Unforgeability:* The cloud server which is the service device for many clients cannot be fully secure. Naturally, the results returned by the server should be validated, while the verification proof for authentication will also be returned. As the significance of this proof, we must make sure it cannot be forged to acquire the real results.

III. Background and Definitions

A. Computation Definitions

Let $\mathbb{G}$, $\mathbb{G}_0$, and $\mathbb{G}'$ be three cyclic groups of multiplication with the same prime order q. Then an asymmetric bilinear map e: $\mathbb{G} \times \mathbb{G}_0 \to \mathbb{G}'$ can be set up with some properties as follows.

1) *Bilinearity:* For any a, $b \in \mathbb{Z}_q^*$, and any g, $h \in \mathbb{G}$, $\mathbb{G}_0$ separately, $e(g^a, h^b) = e(g, h)^{ab}$.
2) *Nondegeneracy:* There exist two data $g \in \mathbb{G}$, $h \in \mathbb{G}$ that satisfy the equation $e(g, h) \neq 1$.
3) *Efficiency:* For any $g \in \mathbb{G}$, $h \in \mathbb{G}_0$, using a polynomial time algorithm related to the security parameter, $e(g, h)$ can be acquired efficiently.

Definition 1 [Co-Computational Diffie–Hellman Problem (co-CDH)]: Let g and g_0 be the generators of $\mathbb{G}$ and $\mathbb{G}_0$, respectively. Randomly chose numbers a and b (a, $b \in \mathbb{Z}_q^*$). For any probabilistic polynomial time (PPT) algorithm adversary, it seems hard to get g^{ab} in non-negligible overhead only using g, g_0, g^a, and g_0^b.

In the scheme, according to asymmetric bilinear map described above, the cloud server will denote the following parameters with a security parameter κ as:

$$\text{params} = \left(q, g_1, g_2, g_0, \mathbb{G}, \mathbb{G}_0, \mathbb{G}', e\right) \leftarrow \mathcal{G}(1^\kappa)$$

where g_1 and g_2 are two generators of $\mathbb{G}$, and g_0 is a generator of $\mathbb{G}_0$. Then, it randomly selects three hash functions with collision-resistant, where $h_1 : \mathbb{Z}_q^* \to \{0, 1\}^*$, $h_2 : \mathbb{Z}_q^* \to \{0, 1\}^*$ and $h_3 : \mathbb{Z}_q^* \to \{0, 1\}^*$.

B. Verifiable Computation

A verifiable computation scheme for outsourcing enables the client to outsource the heavy computation task of functions to any untrusted server, where the client can also verify the correctness of the results returned by the server. Taking efficiency and privacy into consideration, this paper can not only verify the results in public but also protect the input and output both in low overhead.

Let f_1 and f_2 be the function $F(X)$ and $F(X^{(1)}, X^{(2)})$, separately. As outsourcing schemes have some differences in functions f_1 and f_2, our schemes can be described as follows.

Preprocessing(f_1,κ,R)$\to$(SK,PK,VT,V,(β, γ)): In this phase, the client mainly computes verification and encryption information offline. For function f_1, the client first computes the secret key SK for whole scheme together with the public key PK and public verification key V for verification. Besides, the client will also compute verification tag VT for the server to compute verification proof and encryption data (β, γ) for encryption and decryption. Surely, for function f_2, there are some differences. The client should encrypt inputs in advance and generate keys except the public verification key V. The

algorithm can be described as

$$\text{Preprocessing}\left(f_2, \kappa, R, \left(\widetilde{X}^{(1)}, id_1\right), \left(\widetilde{X}^{(2)}, id_2\right)\right) \rightarrow \left(SK, PK, VT, \left(\widetilde{X}^{(1)}, \widetilde{X}^{(2)}\right), \beta, (\gamma_1, \gamma_2)\right).$$

Requesting(f_1, (X, id_x), β) $\rightarrow$ ($\widetilde{X}$): In the phase, the client sends computation request to the cloud server. For function f_1, the input should be encrypted before sending. However, as for the function f_2 the client already encrypts all input matrices in advance, it only generates the public verification key V in this phase. The algorithm can be described as *Requesting*$(f_2, SK, (id_1, id_2)) \rightarrow (V)$.

Computing(f_1, $\widetilde{X}$, VT, id_x) $\rightarrow$ ($\widetilde{Y}$, VP): According to different functions, the cloud server computes the result $\widetilde{Y}$, and uses the verification tag VT to acquire the verification proof VP of $\widetilde{Y}$. For function f_2, the algorithm is

$$\text{Computing}\left(f_2, \left(\widetilde{X}^{(1)}, id_1\right), \left(\widetilde{X}^{(2)}, id_2\right), VT\right) \rightarrow (\widetilde{Y}, VP).$$

Verifying(f_1/f_2, $\widetilde{Y}$, VP, PK, V, id_x/(id_1, id_2)) $\rightarrow$ $\widetilde{Y} \cup \perp$: In this phase, the client can confirm the correctness of results through the third party, according to the verification data VP and V. If the third party proves the validity of $\widetilde{Y}$, the client accepts the result $\widetilde{Y}$, otherwise, refuses.

Decrypting(f_1/f_2, $\widetilde{Y}$, γ/(γ_1, γ_2), id_x/(id_1, id_2)) $\rightarrow$ Y: After conforming the correctness of result computed by the server, the client should decrypt $\widetilde{Y}$ to get final result Y. Using decryption divisors $\gamma/(\gamma_1, \gamma_2)$, this progress can be efficient for function f_1/f_2.

C. Security Definitions

As the verification proof VP is used to validate the result, it is necessary to prove the unforgeability of proof. In the following, we define an experiment $Exp_{\mathcal{A}}^{1^\kappa}$ for the verifiable delegation scheme, which is under an adaptive chosen-message attack.

Step 1: Shown in the phase Preprocessing, the challenger generates the secret key SK and the public key PK with the security parameter κ. Then, the adversary $\mathcal{A}$ can get the public key PK. Naturally, the adversary $\mathcal{A}$ can acquire PK.

Step 2: For one of functions $F(X)$ and $F(X^{(1)}, X^{(2)})$, the adversary $\mathcal{A}$ can query the challenger the verification message of chosen function inputs on the discrete time, adaptively. For function $F(X)$, the adversary $\mathcal{A}$ can obtain the verification tags VT and the public verification key V in Preprocessing algorithm before sending the query about chosen input X to the challenger in *Requesting*. For function $F(X^{(1)}, X^{(2)})$, the challenger sends VT to $\mathcal{A}$ in Preprocessing. Then, after sending the identifications of adaptively chosen inputs to the challenger, $\mathcal{A}$ obtains V from the challenger.

Step 3: In this step, a client requires the adversary $\mathcal{A}$ to evaluate the function $F(X)$ or $F(X^{(1)}, X^{(2)})$, where the inputs required by the client have been chosen by $\mathcal{A}$.

Step 4: Using the information it gets, $\mathcal{A}$ returns a result Y together with the verification proof VP, where $Y \neq F(X)$ or $Y \neq F(X^{(1)}, X^{(2)})$. If Y can satisfy all conditions of verification process and $Y \neq F(X)$ or $Y \neq F(X^{(1)}, X^{(2)})$, the adversary $\mathcal{A}$ wins the experiment.

TABLE I
NOTATIONS

Symbols	Meanings
$X/\widetilde{X}$	the decrypted/encrypted input
$Y/\widetilde{Y}$	the decrypted/encrypted output
ID	the identity of each matrix
SK/PK	the secret/public key
VT	the verification tag
VP	the verification proof
V	the public verification key
α/β	the encryption divisor of inputs/the client
γ	the decryption divisor

Definition 2: We say that a verifiable outsourcing scheme is secure, if the probability that any PPT algorithm adversary $\mathcal{A}$ succeeds in winning the experiment $Exp_{\mathcal{A}}^{1^\kappa}$ is negligible.

IV. Proposed Scheme

In the preliminary conference version (EPP-DMM), this paper presents an outsourcing scheme for matrix multiplication $F(X) = MX$. However, the verification party must verify the each element of the result, so the client should produce the verification tags for each element. As a fact, this step can cause more overhead. Besides, the matrix M is fixed, and the final results depend on X. The structure of matrix multiplication is single, which restricts the application to a great extent. This paper mainly focuses on those two sides, and the new contribution can be described as follows.

1) Avoiding verifying the results by, respectively, proving the correctness of each element, we optimize the scheme to reduce computation overhead. In the new scheme, we only need verify the result using vector as the verification unit. This change improves performance in many sides, especially in computation overhead which reduces from $O(n^2)$ to $O(n)$. When the scheme is applied to a great many calculation of big data, this change, out of question, can bring huge benefits.
2) In the scheme, except function $F(X)$, we expand our scheme to the outsourcing model of another function $F(X^{(1)}, X^{(2)})$. In function $F(X^{(1)}, X^{(2)})$, there are two inputs, which means both two matrices of matrix multiplication are inconstant. When the scheme can be applied to two function models, the applicability is enhanced to a certain extant.

Besides, the main notions are supplied in Table I.

A. Progress-Optimized Outsourcing Scheme for Matrix Multiplication

In this section, we propose a scheme to optimize the verification progress for the function $F(X)$. We can define matrix X as $X = [\vec{x}_1\ \vec{x}_2\ \ldots\ \vec{x}_{n_3}]$, and the output

$$Y = MX = \left[M \cdot \vec{x}_1\ M \cdot \vec{x}_2\ \ldots\ M \cdot \vec{x}_k \ldots\ M \cdot \vec{x}_{n_3}\right] \tag{1}$$

where $M \in \mathbb{Z}_q^{n_1 \times n_2}$, $X \in \mathbb{Z}_q^{n_2 \times n_3}$. The process of verification can be conducted for each operation of $M \cdot \vec{x}_k$. In delegation, the client only sends the computing request to

Algorithm 1 Preprocessing for the Scheme POME

Input: 1^k, M
Output: SK, PK, VT, V

Randomly choose three numbers from $\mathbb{Z}_q^*$, namely $sk_1 = a$, $sk_2 = b$ and $sk_3 = c$
Randomly generate a random row vector $\vec{l} \in \mathbb{Z}_q^{n_1}$
Compute $pk_1 = g_0^a$, $pk_2 = (g_2^{bl_i})_{n_1}$ and $pk_3 = g_0^c$
Compute the row vector $\vec{m} = \vec{l} \cdot M$
for $j = 1$ to n_2 **do**
  Compute $\mu_j = (g_1^{\sum_{i=1}^{n_1} h_1(i,j,sk_1)} g_2^{\sum_{i=1}^{n_1} h_2(i,j,sk_2)+m_j})^{sk_1 sk_2}$, $\mu_j' = (g_1^{\sum_{i=1}^{n_1} h_1(i,j,sk_1)} g_2^{\sum_{i=1}^{n_1} h_2(i,j,sk_2)})^{sk_2}$ and $\mu_j'' = (g_1^{\sum_{i=1}^{n_1} h_1(i,j,sk_1)} g_2^{\sum_{i=1}^{n_1} h_2(i,j,sk_2)})^{sk_2 sk_3}$
end for
Compute $\nu = (g_1^{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} h_1(i,j,sk_1)} g_2^{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} h_2(i,j,sk_2)})^{sk_2}$
The secret key $SK = (sk_1, sk_2, sk_3)$
the public key $PK = (pk_1, pk_2, pk_3)$
The verification tag $VT = (\vec{\mu}, \vec{\mu}', \vec{\mu}'')$
The public verification key $V = \nu$
return SK, PK, VT, V

the cloud server, which means there is no need to compute data in *Requesting* phase. The main algorithm including three phases, i.e., Preprocessing, Computing, *Verifying*, is presented as follows.

1) Preprocessing: As the main task of the client is to generate keys and proof tags in this phase, the generating process of verification data is changed compared with original version. In Algorithm 1, we define a random row vector $\vec{l}$ to deal with the matrix M, where the row vector $\vec{m} = \vec{l} \cdot M$. Then verification progress can be conducted for the equation

$$\vec{l} \cdot \vec{y}_k = \sum_{i=1}^{n_1} l_i y_{i,k} = \sum_{j=1}^{n_2} m_j x_{j,k} = \vec{m} \cdot \vec{x}_k. \tag{2}$$

Accordingly, the public key pk_2 is a vector, where $pk_{2,i} = g_2^{bl_i}$. And the dimension of the verification tag VT can be reduced from 2-D to 1-D, while the public verification key V changes to a number of group $\mathbb{G}$.

2) Requesting: In this phase, the client chooses the input X for the function $F(X)$. Surely, the client will deal with the input X into $\tilde{X}$ before sending it to the server. With all things being prepared, the client sends to the server $\tilde{X}$ as the computation request of function $F(X)$.

3) Computing: When the cloud server receives the computing request from the client, it must compute according to Algorithm 2. The server computes the function $Y = F(X)$ after receiving the input X with the identification id_x from the client. Likewise, the verification proof VP is calculated for verification, where $VP = (\omega, \omega', \omega'')$. The proof VP of different $k \in [1, n_3]$ can be used to verify the different column vectors of the output Y.

4) Verifying: In this phase, the third party verifies the result using Algorithm 3. According to (1), it verifies one column of the result Y at a time. If there exists one column that cannot satisfy any of two verification equations, the result Y must be refused. That is to say, if the result Y is true, each column of Y should content both two verification equations.

Algorithm 2 Computing for the Scheme POMM

Input: M, (X, id_x), VT
Output: Y, VP

$Y = MX$
for $k = 1$ to n_3 **do**
  Compute $\omega_k = \prod_{j=1}^{n_2} (\mu_j)^{x_{j,k}}$, $\omega_k' = \prod_{j=1}^{n_2} (\mu_j')^{x_{j,k}}$ and $\omega_k'' = \prod_{j=1}^{n_2} (\mu_j'')^{x_{j,k}+\sum_{i=1}^{n_1} h_3(i,k,id_x)}$
end for
The verification proof $VP = (\omega, \omega', \omega'')$
return Y, VP

Algorithm 3 Verifying for the Scheme POMM

Input: Y, VP, V, PK
Output: $Y \| \perp$

for $k = 1$ to n_3 **do**
  if

$$e(\omega_k'', g_0) = e\left(\omega_k' \prod_{i=1}^{n_1} \nu^{h_3(i,k,id_x)}, pk_3\right) \tag{3}$$

and

$$e(\omega_k, g_0) = e\left(\omega_k' \prod_{i=1}^{n_1} (pk_{2,i})^{y_{i,k}}, pk_1\right) \tag{4}$$

  then
    Set number t=1
  else
    Set number t=0
    break
  end if
end for
if t=1 **then**
  return Y
else
  return $\perp$
end if

Necessity: In this phase, the algorithm has two verification equation, and both of them are necessary. In (3), it mainly checks the correctness of ω', while it proves the result $\vec{y}_k$ using (4). If we ignore (3), the server can construct the false data $(\vec{y}_k', w')$ easily by the equation $w' = \omega_k' \prod_{i=1}^{n_1} (pk_{2,i})^{y_{i,k}-y_{i,k}'}$.

Correctness: To prove the correctness of scheme POMM, we focus on two sides as follows.

1) If ω_k' is valid, then (3) holds.

According to Algorithm 1, we can get equations $\nu = (g_1^{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} h_1(i,j,sk_1)} g_2^{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} h_2(i,j,sk_2)})^{sk_2} = \prod_{j=1}^{n_2} \mu_j'$ and $\mu_j'' = (g_1^{\sum_{i=1}^{n_1} h_1(i,j,sk_1)} g_2^{\sum_{i=1}^{n_1} h_2(i,j,sk_2)})^{sk_2 sk_3} = (\mu_j')^{sk_3}$.

Then, it is easy to verify the correctness of (3) through equation

$$\begin{aligned} e(\omega''_k, g_0) &= e\left(\prod_{j=1}^{n_2}\left(\mu''_j\right)^{x_{j,k}+\sum_{i=1}^{n_1} h_3(i,k,id_x)}, g_0\right) \\ &= e\left(\prod_{j=1}^{n_2}\left(\left(\mu'_j\right)^{sk_3}\right)^{x_{j,k}+\sum_{i=1}^{n_1} h_3(i,k,id_x)}, g_0\right) \\ &= e\left(\prod_{j=1}^{n_2}\left(\mu'_j\right)^{x_{j,k}} \prod_{j=1}^{n_2}\left(\mu'_j\right)^{\sum_{i=1}^{n_1} h_3(i,k,id_x)}, g_0^{sk_3}\right) \\ &= e\left(\prod_{j=1}^{n_2}\left(\mu'_j\right)^{x_{j,k}} \nu^{\sum_{i=1}^{n_1} h_3(i,k,id_x)}, g_0^{sk_3}\right) \\ &= e\left(\omega'_k \prod_{i=1}^{n_1} \nu^{h_3(i,k,id_x)}, pk_3\right). \end{aligned} \tag{5}$$

2) If $\vec{y}_k$ is valid, then (4) holds.

According to Algorithm 1, we can also acquire the equation $\mu_j = (g_1^{\sum_{i=1}^{n_1} h_1(i,j,sk_1)} g_2^{\sum_{i=1}^{n_1} h_2(i,j,sk_2)+m_j})^{sk_1 sk_2} = (\mu'_j g_2^{bm_j})^{sk_1}$. Then it we can use the following equation to prove the availability of the verification proofs:

$$\begin{aligned} e(\omega_k, g_0) &= e\left(\prod_{j=1}^{n_2}\left(\mu_j\right)^{x_{j,k}}, g_0\right) \\ &= e\left(\prod_{j=1}^{n_2}\left(\left(\mu'_j g_2^{bm_j}\right)^{sk_1}\right)^{x_{j,k}}, g_0\right) \\ &= e\left(\prod_{j=1}^{n_2}\left(\mu'_j\right)^{x_{j,k}} g_2^{b\sum_{j=1}^{n_2} m_j x_{j,k}}, g_0^{sk_1}\right) \\ &= e\left(\left(\prod_{j=1}^{n_2}\left(\mu'_j\right)^{x_{j,k}}\right) g_2^{b\sum_{i=1}^{n_1} l_i y_{i,k}}, g_0^{sk_1}\right) \\ &= e\left(\omega'_k \prod_{i=1}^{n_1}\left(g_2^{bl_i}\right)^{y_{i,k}}, pk_1\right) \\ &= e\left(\omega'_k \prod_{i=1}^{n_1}\left(pk_{2,i}\right)^{y_{i,k}}, pk_1\right). \end{aligned} \tag{6}$$

Flexibility: The algorithm we described above is designed for the verification progress conducted by the third party. It is flexible for the client to conduct the verification progress by itself as follows. If $\vec{y}_k$ is valid, then two equations $\omega''_k = (\omega'_k \prod_{i=1}^{n_1} \nu^{h_3(i,k,id_x)})^c = (\omega'_k \nu^{\sum_{i=1}^{n_1} h_3(i,k,id_x)})^c$ and $\omega_k = (\omega'_k \prod_{i=1}^{n_1} (pk_{2,i})^{y_{i,k}})^a = (\omega'_k g_2^{\sum_{i=1}^{n_1} l_i y_{i,k}})^{ab}$ are hold.

B. Application Function-Expanded Outsourcing Scheme for Matrix Multiplication

In matrix multiplication of function $F(X)$, the matrix M is constant while only the input X changes each time. This restricts the application to a large extent. In this scheme, we extent the original scheme and apply it to a new function $F(X^{(1)}, X^{(2)})$, where $Y = F(X^{(1)}, X^{(2)}) = X^{(1)}X^{(2)}$ ($X^{(1)} \in D^{(1)}$, $X^{(2)} \in D^{(2)}$). Likewise, we conduct the verification phase for each operation of $X^{(1)} \cdot \vec{x}_k^{(2)}$, similar to the scheme POMM. As the input $X^{(1)}$ and $X^{(2)}$ are decided in the phase *Requesting*, this scheme involves four phases, i.e., Preprocessing, *Requesting*, Computing, and *Verifying*, involved.

Algorithm 4 Preprocessing for the Scheme AFEMM

Input: 1^k, $(X^{(1)}, id_1)$, $(X^{(2)}, id_2)$
Output: SK, PK, VT

1: Randomly choose four numbers from $\mathbb{Z}_q^*$, namely $sk_1 = a$, $sk_2 = b$, $sk_3 = c$ and $sk_4 = d$
2: Randomly generate a random row vector $\vec{l} \in \mathbb{Z}_q^{n_1}$
3: Compute $pk_1 = g_0^a$, $pk_2 = (g_2^{bl_i})_{n_1}$ and $pk_3 = g_0^c$
4: Compute the row vector $\vec{\chi} = \vec{l} \cdot X^{(1)}$
5: **for** $j = 1$ to n_2 **do**
6: Compute
$\mu_j^{(1)} = (g_1^{\sum_{i=1}^{n_1} h_1(i,j||id_1,sk_1)} g_2^{\sum_{i=1}^{n_1} h_2(i,j||id_1,sk_2)+\chi_j})^{sk_1 sk_2}$,
$\mu_j'^{(1)} = (g_1^{\sum_{i=1}^{n_1} h_1(i,j||id_1,sk_1)+d} g_2^{\sum_{i=1}^{n_1} h_2(i,j||id_1,sk_2)})^{sk_2}$,
$\mu_j''^{(1)} = (g_1^{\sum_{i=1}^{n_1} h_1(i,j||id_1,sk_1)} g_2^{\sum_{i=1}^{n_1} h_2(i,j||id_1,sk_2)})^{sk_2 sk_3}$
for matrix $X^{(1)}$
7: Compute
$\mu_j^{(2)} = (g_1^{\sum_{k=1}^{n_3} h_1(k,j||id_2,sk_1)+d} g_2^{\sum_{k=1}^{n_3} h_2(k,j||id_2,sk_2)})^{sk_1 sk_2}$,
$\mu_j'^{(2)} = (g_1^{\sum_{k=1}^{n_3} h_1(k,j||id_2,sk_1)} g_2^{\sum_{k=1}^{n_3} h_2(k,j||id_2,sk_2)})^{sk_2}$ and
$\mu_j''^{(2)} = (g_1^{\sum_{k=1}^{n_3} h_1(k,j||id_2,sk_1)+d} g_2^{\sum_{k=1}^{n_3} h_2(k,j||id_2,sk_2)})^{sk_2 sk_3}$
for matrix $X^{(2)}$
8: **end for**
9: The secret key $SK = (sk_1, sk_2, sk_3, sk_4)$
the public key $PK = (pk_1, pk_2, pk_3)$
10: The verification tag $VT^{(1)} = (\vec{\mu}^{(1)}, \vec{\mu}'^{(1)}, \vec{\mu}''^{(1)})$ of $X^{(1)}$
The verification tag $VT^{(2)} = (\vec{\mu}^{(2)}, \vec{\mu}'^{(2)}, \vec{\mu}''^{(2)})$ of $X^{(2)}$
Let $VT = (VT^{(1)}, VT^{(2)})$
11: **return** SK, PK, VT

1) Preprocessing: Similar to the scheme POMM, we acquire the row vector $\vec{\chi}$ ($\vec{\chi} = \vec{l} \cdot X^{(1)}$), and (7) can be used to verification

$$\vec{l} \cdot \vec{y}_k = \sum_{i=1}^{n_1} l_i y_{i,k} = \sum_{j=1}^{n_2} \chi_j x_{j,k}^{(2)} = \vec{\chi} \cdot \vec{x}_k^{(2)}. \tag{7}$$

In the function $F(X^{(1)}, X^{(2)})$, every matrix of database $D^{(1)}$ and $D^{(2)}$ should be preprocessed before being sent to the cloud server. As shown in Algorithm 4, we only consider one time operation for $(X^{(1)}, X^{(2)})$. In this model, each matrix has its own identification, where id_1 and id_2 belong to $X^{(1)}$ and $X^{(2)}$, separately. Besides, we produce corresponding verification tag VT for inputs $(X^{(1)}, X^{(2)})$, and the secret key SK changes to four numbers. After all preprocessing for $D^{(1)}$ and $D^{(2)}$, the data, e.g., VT, $(X^{(1)}, id_1)$ and $(X^{(2)}, id_2)$, should be sent to the server in advance, where the client only stores the *id* of each matrix to stand for the matrices in database.

2) Requesting: In this phase, the client sends the computing request to the cloud server, which means the matrices $X^{(1)}$ and

Algorithm 5 Computing for the Scheme AFEMM

Input: $(X^{(1)}, id_1)$, $(X^{(2)}, id_2)$, VT
Output: Y, VP
1: $Y = X^{(1)}X^{(2)}$
2: **for** $k = 1$ to n_3 **do**
3: Compute
$\omega_k = \prod_{j=1}^{n_2} (\mu_j^{(1)}\mu_j^{(2)})^{x_{j,k}^{(2)}}$, $\omega'_k = \prod_{j=1}^{n_2} (\mu_j'^{(1)}\mu_j'^{(2)})^{x_{j,k}^{(2)}}$
and $\omega''_k = \prod_{j=1}^{n_2} (\mu_j''^{(1)}\mu_j''^{(2)})^{x_{j,k}^{(2)}+\sum_{i=1}^{n_1} h_3(i,k,id_1||id_2)}$
4: **end for**
5: The verification proof $VP = (\omega, \omega', \omega'')$
6: **return** Y, VP

$X^{(2)}$ should be determined for the function $F(X^{(1)}, X^{(2)})$. Then, we obtain summing values of hash values, where

$$\begin{cases} H_{11} = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} h_1(i,j||id_1, sk_1),\ H_{12} = \sum_{k=1}^{n_3}\sum_{j=1}^{n_2} h_1(i,j||id_2, sk_1) \\ H_{21} = \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} h_2(i,j||id_1, sk_2),\ H_{22} = \sum_{k=1}^{n_3}\sum_{j=1}^{n_2} h_2(i,j||id_2, sk_2). \end{cases}$$

Let the verification key V as

$$V = \nu = \left(g_1^{H_{11}+H_{12}+dn_2} g_2^{H_{21}+H_{22}}\right)^{sk_2}. \tag{8}$$

The client only sends the cloud server (id_1, id_2) as the computing request, and publishes the verification key V.

3) Computing: According to the request (id_1, id_2) from the client, the cloud server can acquire the related information from its database, which includes a tuple (Id, Y', Vp). The tuple records the data of precious computing task, where Id is the set of elements (id', id''), Y', and Vp are the sets of the corresponding calculated data.

If (id_1, id_2) requested by the client belongs to Id, the server can directly return corresponding result $Y \in Y'$ and verification proof $VP \in Vp$. Otherwise, the server calculates the function $F(X^{(1)}, X^{(2)})$ together with the verification proof VP for the client. After computing, the cloud server not only sends Y and VP to the client, but also updates $((id_1, id_2), Y, VP)$ to the tuple (Id, Y', Vp).

4) Verifying: Through the function is changed to $F(X^{(1)}, X^{(2)})$, the verification process is similar. Equation (4) is also adopted to verify the result Y, while another equation is a little different. The changed identification information makes the input of hash function h_3 become $id_1||id_2$, and the another verification equation can be expressed as

$$e\left(\omega''_k, g_0\right) = e\left(\omega'_k \prod_{i=1}^{n_1} \nu^{h_3(i,k,id_1||id_2)}, pk_3\right). \tag{9}$$

If the return is Y, the result Y proves to be true, otherwise refuse the result.

In this phase, it also has properties of necessity, correctness, and flexibility. Obviously, the property necessity is tenable. Just like the scheme POMM, the scheme has the flexibility when changing the input id_x to $id_1||id_2$.

Correctness: To prove the correctness of this scheme, we compare it with the scheme POMM.

Although the verification proofs $VT^{(1)}$ and $VT^{(2)}$ are related to two input matrices, there are many places in common to prove the correctness of (4) and (9).

Preprocessing:
1: Randomly chose n_2 numbers of $\mathbb{Z}_q$ to acquire the encryption divisor $\beta = (b_j)_{n_2}$
2: **for** $i = 1$ to n_1 **do**
3: Compute $r_i = \sum_{j=1}^{n_2} m_{i,j} b_j$
4: **end for**
5: The decryption divisor $\gamma = (r_i)_{n_1}$
6: **return** (β, γ)

Requesting:
1: Set another encryption divisor, the vector $\alpha = (h_1(i, id_x, sk_1))_{n_1}$
2: Compute

$$\widetilde{X} = X + \beta\alpha^T \tag{10}$$

3: **return** $\widetilde{X}$

Decrypting:
1: Compute the vector $\alpha = (h_1(i, id_x, sk_1))_{n_1}$ again.
2: Compute

$$Y = \widetilde{Y} - \gamma\alpha^T \tag{11}$$

3: **return** Y

Fig. 2. Encryption scheme for POMM.

According to Algorithm 4, we can also get the equation $\nu = ((\prod_{j=1}^{n_2} g_1^d) g_1^{H_{11}+H_{12}} g_2^{H_{21}+H_{22}})^{sk_2} = \prod_{j=1}^{n_2}(\mu_j'^{(1)}\mu_j'^{(2)})$ and $\mu_j''^{(1)}\mu_j''^{(2)} = (\mu_j'^{(1)}\mu_j'^{(2)})^{sk_3}$. Then (9) proves to be true, like (5), when putting two equations above into it.

According to Algorithm 4, we can acquire equations $\mu_j^{(1)}\mu_j^{(2)} = (\mu_j'^{(1)}\mu_j'^{(2)} g_2^{bm_j})^{sk_1}$. Putting this equation into (4), the equation is tenable.

C. Encryption Scheme

As inputs may expose the sensitive information of the client, the client must preserve the privacy of inputs after sending them to the cloud server [43]. In this section, we propose an encryption scheme to ensure the security.

For each matrix of inputs, we encrypt it into $\widetilde{X}$. Then two functions can be described as $\widetilde{Y} = F(\widetilde{X})$ and $\widetilde{Y} = F(\widetilde{X}^{(1)}, \widetilde{X}^{(2)})$. Thanks to the different functions for application, the encryption and decryption courses of two schemes are distinguishing in some way.

1) Encryption Scheme for POMM: As shown in Fig. 2, we use (10) to encrypt the input X in *Requesting* phase, while in phase Preprocessing, the client will construct encryption and decryption divisors in advance. In the *Requesting* phase, the client sends the request to the server $\widetilde{X}$ rather than X. And (β, γ) is kept in secret by the client. Another encryption divisor α, related to the input, is obtained through hash function when needed each time. As $M\beta = (\sum_{j=1}^{n_2} m_{i,j} b_j)_{n_1} = \gamma$, we give the demonstration to prove the correctness of (11) as

$$Y = MX = M\left(\widetilde{X} - \beta\alpha^T\right) = \widetilde{Y} - (M\beta)\alpha^T = \widetilde{Y} - \gamma\alpha^T. \tag{12}$$

2) Encryption Scheme for AFEMM: In Fig. 3, we use (13) to protect the inputs, and get the final result Y by (14). After encrypting all the matrices of database, the client only sends the blind inputs, e.g., $(\widetilde{X}^{(1)}, \widetilde{X}^{(2)})$, to the cloud server. Because α_1 and α_2 are constructed by hash functions, the client only needs to save β and (γ_1, γ_2). As $X^{(1)}\beta =$

Preprocessing:

Randomly chose n_2 numbers of $\mathbb{Z}_q$ to acquire the encryption divisor $\beta = (b_j)_{n_2}$

Set other encryption divisors $\alpha_1 = (h_1(i, id_1, sk_1))_{n_1}$ and $\alpha_2 = (h_2(i, id_2, sk_2))_{n_1}$ for inputs $X^{(1)}$ and $X^{(2)}$, respectively.

Compute

$$\begin{cases} \widetilde{X}^{(1)} = X^{(1)} + \alpha_1 \beta^T \\ \widetilde{X}^{(2)} = X^{(2)} + \beta \alpha_2^T \end{cases} \tag{13}$$

for $i = 1$ to n_1 **do**

  Compute $r_i = \sum_{j=1}^{n_2} x_{i,j}^{(1)} b_j$

end for

for $k = 1$ to n_3 **do**

  Compute $t_k = \sum_{j=1}^{n_2} b_j \widetilde{x}_{i,j}^{(2)}$

end for

The decryption divisors $\gamma_1 = (r_i)_{n_1}$ and $\gamma_2 = (t_i)_{n_3}$

return $(\widetilde{X}^{(1)}, \widetilde{X}^{(2)})$, β, (γ_1, γ_2)

Decrypting:

Compute $\alpha_1 = (h_1(i, id_1, sk_1))_{n_1}$ and $\alpha_2 = (h_2(i, id_2, sk_2))_{n_1}$ again.

Compute

$$Y = \widetilde{Y} - \gamma_1 {\alpha_2}^T - \alpha_1 \gamma_2^T \tag{14}$$

return Y

Fig. 3. Encryption scheme for AFEMM.

$(\sum_{j=1}^{n_2} x_{i,j}^{(1)} b_j)_{n_1} = \gamma_1$ and $\beta \widetilde{X}^{(2)} = (\sum_{j=1}^{n_2} b_j \widetilde{x}_{i,j}^{(2)})_{1 \times n_1} = \gamma_2^T$, we can prove the correctness of (14) as follows:

$$\begin{aligned} Y = X^{(1)} X^{(2)} &= X^{(1)} \left(\widetilde{X}^{(2)} - \beta \alpha_2^T \right) \\ &= \left(\widetilde{X}^{(1)} - \alpha_1 \beta^T \right) \widetilde{X}^{(2)} - X^{(1)} \beta \alpha_2^T \\ &= \widetilde{X}^{(1)} \widetilde{X}^{(2)} - \alpha_1 \left(\beta^T \widetilde{X}^{(2)} \right) - \left(X^{(1)} \beta \right) \alpha_2^T \\ &= \widetilde{Y} - \gamma_1 {\alpha_2}^T - \alpha_1 \gamma_2^T. \end{aligned} \tag{15}$$

V. Security Analysis

A. Data Privacy

In schemes, the client will encrypt the inputs before uploading to the server. As the inputs of functions $F(X)$ and $F(X^{(2)}, X^{(2)})$ are protected in same theory, the input X of function $F(X)$ is disposed as $\widetilde{X} = X + \beta \alpha^T$ in (10), for instance. An adversary knows nothing about β and sk_1, and the values of α change with the different input, where the vectors β is randomly chosen, α is composed by the hash function related to the secret key sk_1 and input id. It is hard for the adversary to recover the original input X in consideration of the indistinguishability of matrix. In the similar way, even if the adversary possesses the result $\widetilde{Y}$, it also cannot obtains anything matters from it.

B. Proof Unforgeability

To prove the unforgeability of proof in our scheme, we conduct the following game in the random model for different functions. Besides, in this section, we consider the process of publicly verification as an example.

Theorem 1: According to the assumption co-CDH, the publicly verifiable outsourcing scheme POMM is secure in the random model for successfully against the adaptive chosen-message attack.

Proof: According to the scheme POMM, the verification process of function $Y = F(X)$ can be treated as the verification process of function $\vec{y}_k = F(\vec{x}_k)$. As there are two equations for verification, we prove them, respectively.

1) If ω_k'' can pass the verification (3), then ω_k' is valid.

Suppose ω_k' is one of the verification proof returned by the adversary $\mathcal{A}$ and $\omega_k'^*$ is the real one. Both ω_k' and $\omega_k'^*$ can satisfy (3), where $\omega_k' \neq \omega_k'^*$. Then we can get the equation as

$$\begin{aligned} \omega_k'' &= \left(\omega_k' \prod_{i=1}^{n_1} \nu^{h_3(i,k,id_x)} \right)^c \\ &= \left(\frac{\omega_k'}{\omega_k'^*} \cdot \omega_k'^* \prod_{i=1}^{n_1} \nu^{h_3(i,k,id_x)} \right)^c \\ &= \left(\frac{\omega_k'}{\omega_k'^*} \right)^c \cdot \omega_k''. \end{aligned}$$

Since $\omega_k' \neq \omega_k'^*$ and $c \neq 0$, $\omega_k'' = (\omega_k'/\omega_k'^*)^c \cdot \omega_k''$ is false obviously. Hence, if ω_k'' can pass the verification (3), $\omega_k' = \omega_k'^*$.

2) If ω_k' is valid and ω_k can pass the verification (4), then $\vec{y}_k$ is valid.

Suppose that the probability of the adversary $\mathcal{A}$ winning in the experiment $Exp_{\mathcal{A}}^{1^\kappa}$ is negligible. There, we construct an adversary $\mathcal{A}^*$ to solve the problem co-CDH collaborating with $\mathcal{A}$. That is, possessing a co-CDH parameters $(g_0, g_2, g_0^a, g_2^{bl_t})$, the adversary $\mathcal{A}^*$ can calculate $g_2^{abl_t}$ in nonignorable probability. In this part, $\mathcal{A}^*$ randomly chooses a public key $g_2^{bl_t}$ $(t \in n_1)$ from $pk_2 = (g_2^{bl_i})_{n_1}$.

Step 1: As the adversary $\mathcal{A}^*$ knows the parameters $(g_0, g_1, g_2, pk_1, pk_2, pk_3)$, it redefines parameters g_1^*, pk_2^* and pk_3^*. $\mathcal{A}^*$ constructs $g_1^* = {g_2}^{-\varepsilon^{-1}}$ and $pk_3^* = g_0^{c^*}$, where randomly chosen numbers $\varepsilon, c^* \in \mathbb{Z}_q^*$. After generating a random row vector $\vec{l}^* \in \mathbb{Z}_q^{n_1}$, $\mathcal{A}^*$ computes $pk_2^* = (g_2^{bl_t l_i^*})_{n_1}$. Finally, new system parameters g_1^*, pk_2^* and pk_3^* are updated to $\mathcal{A}$.

Step 2: The adversary $\mathcal{A}$ can query the adversary $\mathcal{A}^*$ the verification message of chosen inputs on the discrete time, adaptively. It requests the calculation of function $F(\vec{x}_k) = M \cdot \vec{x}_k$.

The adversary $\mathcal{A}^*$ computes the row vector $\vec{m}^* = \vec{l}^* \cdot M$ and chooses some random numbers from $\mathbb{Z}_q^*$ to construct an array $Z = (z_j)_{n_2}$. Let $\vec{m} = l_t \vec{m}^*$, $\sum_{i=1}^{n_1} h_1(i, j, sk_1) = \varepsilon z_j$ and $\sum_{i=1}^{n_1} h_2(i, j, sk_2) = -m_j + z_j$. Then it can recompute verification tags $VT = (\mu, \mu', \mu'')$, of which elements are calculated as

$$\begin{cases} \mu_j = \left(\left(g_1^* \right)^{\varepsilon z_j} g_2^{(-m_j + z_j) + m_j} \right)^{ab} = 1 \\ \mu_j' = \left(\left(g_1^* \right)^{\varepsilon z_j} g_2^{-m_j + z_j} \right)^{b} = g_2^{-bl_t m_j^*} \\ \mu_j'' = \left(\left(g_1^* \right)^{\varepsilon z_j} g_2^{-m_j + z_j} \right)^{bc^*} = g_2^{-bl_t m_j^* c^*}. \end{cases}$$

Besides, taking all defined data into the equation of V, $\mathcal{A}^*$ also gets the public verification key V. Surely, VT will be sent to $\mathcal{A}$. Because VT and V are calculated strictly following equations

in scheme POMM, the verification equations are tenable if the results and proofs returned by the server are correct.

Step 3: Next, $\mathcal{A}^*$ requests the adversary $\mathcal{A}$ to evaluate the result of function $F(X)$ and the corresponding verification proof VP.

Step 4: Returned by $\mathcal{A}$, the result Y can be verified using the verification proof VP. According to the experiment $Exp_{\mathcal{A}}^{1^\kappa}$ we supposed above, Y can pass all equations of the verification process, while $Y \neq F(X)$. Let $Y^* = F(X)$ be the real result. Then we can acquire the equation as

$$\begin{aligned}\omega &= \left(\omega' \prod_{i=1}^{n_1} pk_2^{y_{i,k}}\right)^a \\ &= \left(\prod_{j=1}^{n_2} (\mu_j)^{x_{j,k}} g_2^{\sum_{i=1}^{n_1} bl_t l_i y_{i,k}}\right)^a \\ &= g_2^{-\sum_{j=1}^{n_2} bl_t m_j^* x_{j,k}} g_2^{\sum_{i=1}^{n_1} bl_t l_i y_{i,k}}\Big)^a \\ &= \left(g_2^{-\sum_{i=1}^{n_1} l_i y_{i,k}^* + \sum_{i=1}^{n_1} l_i y_{i,k}}\right)^{abl_t} \\ &= g_2^{abl_t \sum_{i=1}^{n_1} l_i \left(y_{i,k} - y_{i,k}^*\right)}. \end{aligned} \tag{16}$$

The adversary $\mathcal{A}^*$ can acquire the equation $g_2^{abl_t} = \omega^{\left(\sum_{i=1}^{n_1} l_i (y_{i,k} - y_{i,k}^*)\right)^{-1}}$ when $\sum_{i=1}^{n_1} l_i (y_{i,k} - y_{i,k}^*) \neq 0$. As $\mathcal{A}^*$ has a knowledge of $(g_0, g_2, g_0^a, g_2^{bl_t})$, it can obtain $g_2^{abl_t}$ from $\mathcal{A}$. Surely, while $y_{i,k} \neq y_{i,k}^*$, if $\sum_{i=1}^{n_1} l_i (y_{i,k} - y_{i,k}^*) = 0$, $\mathcal{A}^*$ should reselect new vector $\vec{l}^*$ to conduct this challenge game.

In the conclusion, in the random model, the publicly verifiable outsourcing scheme POMM can against the adaptive chosen-message attack under the assumption co-CDH.

Theorem 2: According to the assumption co-CDH, the publicly verifiable outsourcing scheme AFEMM is secure in the random model for successfully against the adaptive chosen-message attack.

Proof: Similar to the scheme POMM, the verification process of function $Y = F(X^{(1)}, X^{(2)})$ can also be treated as the verification process of function $\vec{y}_k = F(X^{(1)}, \vec{x}_k^{(2)})$ in AFEMM. Then we prove the scheme in the same way as follows.

1) If ω_k'' can pass the verification equation (3), then ω_k' is valid. As only hash function of (9) is different from (3), the way of proving ω_k' can refer to above process.
2) If ω_k' is valid and ω_k can pass the verification equation (4), then $\vec{y}_k$ is valid.

Same as the demonstration of POMM, in this part, the adversary $\mathcal{A}^*$ also has the knowledge of the co-CDH parameters $(g_0, g_2, g_0^a, g_2^{bl_t})$.

Step 1: As the adversary $\mathcal{A}^*$ knows the parameters $(g_0, g_1, g_2, pk_1, pk_2, pk_3)$, it should also redefine parameters $g_1^* = g_2^{-\varepsilon^{-1}}$, $pk_2^* = (g_2^{bl_t l_i^*})_{n_1}$ and $pk_3^* = g_0^{c^*}$. $\mathcal{A}^*$ constructs g_1^* and pk_3^*, where randomly chosen numbers $\varepsilon, c^* \in \mathbb{Z}_q^*$ and a random row vector $\vec{l}^* \in \mathbb{Z}_q^{n_1}$. Finally, new system parameters g_1^*, pk_2^*, and pk_3^* are updated to $\mathcal{A}$.

Step 2: The adversary $\mathcal{A}$ queries the adversary $\mathcal{A}^*$ the verification information of chosen inputs on the discrete time, adaptively. It requests the calculation of function $F(X^{(1)}, X^{(2)}) = X^{(1)} X^{(2)}$.

TABLE II
NOTATIONS

Symbols	Meanings
T_z/T_g	an operation time of multiplication in $\mathbb{Z}_q/\mathbb{G}$
T_i	an operation time of producing inversion matrix
$T_e/T_{e_0}/T'$	an operation time of exponentiation in $\mathbb{G}/\mathbb{G}_0/\mathbb{G}'$
T_b	an operation time of bilinear pairing
$n_1/n_2/n_3$	the dimension of matrixes
N	the number of request times
N_1/N_2	the number of matrixes in $D^{(1)}/D^{(2)}$
l	opposite maximum of $\mathbb{Z}_q$

The adversary $\mathcal{A}^*$ computes the row vector $\vec{\chi}^* = \vec{l}^* \cdot X^{(1)}$. Besides, it chooses some random numbers from $\mathbb{Z}_q^*$ to construct an array $Z = (z_j)_{n_2}$ together with a new randomly chosen number b^* of $\mathbb{Z}_q^*$. It defines $\vec{\chi} = l_t \vec{\chi}^*$ and $d = \varepsilon d^* l_t$. Let $\sum_{i=1}^{n_1} h_1(i, j || id_1, sk_1) = -\varepsilon d^* l_t$, $\sum_{k=1}^{n_3} h_1(k, j || id_2, sk_1) = \varepsilon z_j$, $\sum_{i=1}^{n_1} h_2(i, j || id_1, sk_2) = -\chi_j - d^* l_t$ and $\sum_{k=1}^{n_3} h_2(k, j || id_2, sk_2) = z_j + d^* l_t$. Then it can recompute verification tags $VT^{(1)} = (\vec{\mu}^{(1)}, \vec{\mu}'^{(1)}, \vec{\mu}''^{(1)})$ and $VT^{(2)} = (\vec{\mu}^{(2)}, \vec{\mu}'^{(2)}, \vec{\mu}''^{(2)})$ for matrices $X^{(1)}$ and $X^{(2)}$, of which elements are calculated as

$$\begin{cases} \mu_j^{(1)} = \mu_j^{(2)} = 1 \\ \mu_j'^{(1)} = g_2^{-bl_t\left(\chi_j^* + b^*\right)}, \ \mu_j'^{(2)} = g_2^{bl_t b^*} \\ \mu_j''^{(1)} = g_2^{-bl_t \chi_j^* c^*}, \ \mu_j''^{(2)} = 1. \end{cases}$$

Similarly, $\mathcal{A}^*$ also gets the public verification key V. Because VT and V are calculated strictly following equations in scheme AFEMM, the verification equations are tenable if the results and proofs returned by the server are correct. Finally, $VT = (VT^{(1)}, VT^{(2)})$ will be sent to $\mathcal{A}$.

Step 3: Next, $\mathcal{A}^*$ requests the adversary $\mathcal{A}$ to evaluate the result of function $F(X^{(2)}, X^{(2)})$ and the corresponding verification proof VP.

Step 4: Returned by $\mathcal{A}$, the result Y can be verified using the verification proof VP. According to the experiment $Exp_{\mathcal{A}}^{1^\kappa}$ we supposed above, Y can pass all equations of the verification process, while $Y \neq F(X^{(2)}, X^{(2)})$. Let $Y^* = F(X^{(2)}, X^{(2)})$ be the real result. Then we can also obtain the equation

$$\omega = g_2^{abl_t \sum_{i=1}^{n_1} l_i \left(y_{i,k} - y_{i,k}^*\right)}.$$

Like step 4 of POMM, while $\mathcal{A}^*$ has a knowledge of $(g_0, g_2, g_0^a, g_2^{bl_t})$, it can obtain $g_2^{abl_t}$ from $\mathcal{A}$, eventually.

In the conclusion, in the random model, the publicly verifiable outsourcing scheme AFEMM can against the adaptive chosen-message attack under the assumption co-CDH.

VI. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our schemes comparing with other schemes. Due to the two different function models, we presents the performance in two parts. Besides, involved symbols are shown in Table II.

TABLE III
COMPARISON OF FUNCTIONALITY FOR POMM

	Zhang's [39]	Li's [40]	*EPP-DMM*	*POMM*
Self-Verification	No	No	Yes	Yes
Public Verification	Yes	Yes	Yes	Yes
Data Encryption	Yes	Yes	Yes	Yes

TABLE IV
COMPARISON OF COMMUNICATION OVERHEAD FROM CLIENT TO SERVER FOR POMM

	Client to Server	
	Preprocessing	*Requesting*
Zhang's [39]	$n_1n_2\|\mathbb{Z}_q\| + n_1n_2\|\mathbb{G}\|$	$Nn_2n_3\|\mathbb{G}\|$
Li's [40]	$n_1n_2\|\mathbb{Z}_q\| + n_1n_2\|\mathbb{G}\|$	$Nn_3n_2\|\mathbb{Z}_q\|$
EPP-DMM	$n_1n_2\|\mathbb{Z}_q\| + n_1n_2\|\mathbb{G}\|$	$Nn_2n_3\|\mathbb{Z}_q\|$
POMM	$n_1n_2\|\mathbb{Z}_q\| + n_2\|\mathbb{G}\|$	$Nn_2n_3\|\mathbb{Z}_q\|$

A. Performance Evaluation for POMM

For function model of $F(X)$, we compare our scheme POMM with other similar schemes, Zhang's [39], Li's [40] and EPP-DMM, where Zhang's [39] and Li's [40] are based on the algebraic PRFs with closed form efficiency, and EPP-DMM is the initial version of POMM. In the following, we conduct the evaluation in terms of functionality as well as communication, computation and storage overhead.

1) Functionality: Shown in Table III, all the schemes can achieve public verification while only our schemes EPP-DMM and POMM can verify the results in an efficient way by the client itself. As Zhang's [39], which extends the scheme of Fiore's [29], designs the encryption scheme for inputs all the schemes have the mechanism of data protection. We can see our schemes are more flexible in functionality.

2) Communication Overhead: In scheme POMM, the server will send the verification proofs and the results to the verification party, and the verification party only returns the correct results to the client. Therefore, we only consider the communication overhead of the client sending to the server in process Preprocessing and *Requesting*, and obtain Table IV. In Preprocessing, the client will send matrix M and verification tags VT for one time in advance in amortized model. In the Table IV, we can see that the communication overhead of POMM is lower than others, because the size of verification tags is $n_2|\mathbb{G}|$ while the size of others is $n_1n_2|\mathbb{G}|$. Considering the online process *Requesting*, the communication overhead is determined by the computation request times N. In all those schemes, the client only sends the input data to the server, so the communication overhead of client to server is $n_2n_3|\mathbb{Z}_q|$ for one time. Therefore, our new scheme POMM is efficient in communication overhead.

3) Computation Overhead: For the low ability of client in computation, the computation overhead of the client is the main factor to improve the efficiency of outsourcing scheme. In this part, we compare our scheme POMM with other schemes of phases Preprocessing, *Requesting*, and *Decrypting*, in Table V. In amortized model, the preprocessing phase is conducted for one time. Hence, decisive phases in computation cost are the other two phases.

In Preprocessing process, if the computing times is not large, this process may determine the computation overhead. According to the Table V, the computation overhead of Li's [40] is related to the opposite maximum l, and the time complexity is $O(n^3)$ while the time complexity of others is only $O(n^2)$. If the matrix involved is large, Li's [40] is obviously in low efficiency. Besides, the operation times of exponentiation in $\mathbb{G}'$ and bilinear pairing, i.e., T' and T_b, are larger than other operations in those schemes. In this process, our scheme POMM avoids using those two kinds of operation, which reduces the computation overhead to a great extent in Preprocessing. Moreover, as we optimize the process of verifying in scheme POMM, verification tags is also changed in Preprocessing compared with scheme EPP-DMM. Hence, time cost $O(n)$ of POMM to generate the verification tags is lower than time cost $O(n^2)$ of EPP-DMM.

In *Requesting*, schemes EPP-DMM and POMM should only encrypt the inputs, while other two schemes [39], [40] also generate the verification tags related to the inputs. In *Decrypting*, schemes EPP-DMM and POMM both take $n_1n_3T_z$ time cost in decryption. Especially, the overhead of Zhang's is $O(e^x)$, where the time complexity of exponent $O(e^x)$ is larger than the time cost of $O(n^2)$. Besides, the computation overhead of Li's in decryption can be ignored. Taking those two phases into consideration together, EPP-DMM and POMM with overhead $(n_1n_3 + n_2n_3)T_z$ are more efficient than others of which the overhead are $n_2n_3(T_g + T_e + T_b) + T_{e0} + O(e^x)$ in [39] and $(3n_2n_3 + n_3 + 1)T_z + T_e + T_b$ in [40].

In the conclusion, the scheme POMM is most efficient among four schemes in computation overhead of the client. To prove this inference both in theory and reality, we process the experiment on a computing machine which has the 2.5GHz-processor and 4GB memory. Because of obvious huge computation overhead in [39], we only compare other three schemes to research the computation overhead influenced by the request times N. There, we utilize time consumption to simulate the computation overhead.

Let all the matrices involved be the square matrices and the opposite maximum number l in Li's be $l = 2n$, where $n_1 = n_2 = n_3 = n = 10000$. As shown in Fig. 4, with the raise of request time N the time consumption of Li's scheme is higher than EPP-DMM and POMM obviously. Surely, the bigger l is, the more inefficient Li's scheme is. Furthermore, though the line's growth trends of EPP-DMM and POMM in figure are similar, POMM is efficient than EPP-DMM when N is small oppositely. Obviously, our scheme POMM can achieve high-efficiency no matter how much the request times N is.

4) Storage Overhead: Considering the storage overhead of the client, it stores the secret key together with verification and encryption information. As shown in Table VI, in the schemes of Li's and EPP-DMM the storage overheads are related to the number of request N while the storage overheads of Zhang's and POMM are fixed, namely, the client only store the data produced in the preprocessing phase. In our scheme, many parameters can be computed once used, especially for the hash values. Obviously, the overhead of ours, i.e., $(2n_1+n_2+3)|\mathbb{Z}_q|$, is lower than others, when the stored data of ours only involves the type of $\mathbb{Z}_q$.

TABLE V
COMPARISON OF COMPUTATION OVERHEAD FOR POMM

Phase	*Preprocessing*	*Requesting* (Multiple: N)	*Decrypting* (Multiple: N)
Zhang's [39]	$(3n_1n_2+2)T_z+3n_1n_2T_g+(n_1n_2+3)T_e$	$n_2n_3(T_g+T_e+T_b)+T_{e0}$	$O(e^x)$
Li's [40]	$(n_1n_2l+n_1n_2+n_2)T_z$ $+(n_1n_2+1)T_e+n_1l(T_{e_0}+T')+T_b$	$(3n_2n_3+n_3+1)T_z+T_e+T_b$	$-$
EPP-DMM	$3n_1n_2T_z+(n_1+2n_1n_2)T_g$ $+(6n_1n_2+3n_1+1)T_e+2T_{e_0}$	$n_2n_3T_z$	$n_1n_3T_z$
POMM	$(2n_1n_2+2)T_z+(2n_2+1)T_g$ $+(n_1+6n_2+4)T_e+2T_{e_0}$	$n_2n_3T_z$	$n_1n_3T_z$

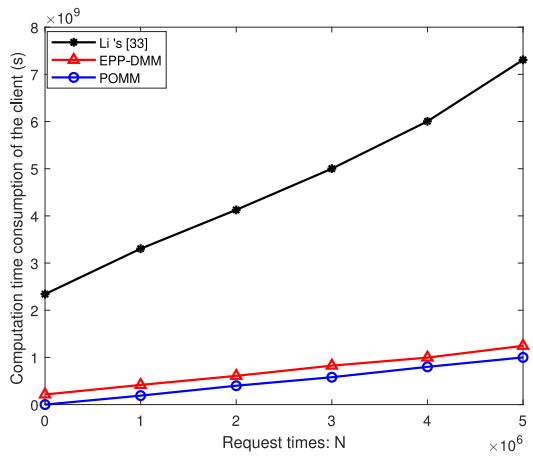


Fig. 4. Computation time consumption of the client.

TABLE VI
COMPARISON OF STORAGE OVERHEAD FOR POMM

	Preprocessing	*Requesting*
Zhang's [39]	$(2n_1+3)\|\mathbb{Z}_q\|+2n_2\|\mathbb{G}\|$	$-$
Li's [40]	$(3n_1l+n_2+1)\|\mathbb{Z}_q\|$	$Nn_2n_3\|\mathbb{Z}_q\|$
EPP-DMM	$(2n_1+n_2+3)\|\mathbb{Z}_q\|$	$N(n_3+1)\|\mathbb{Z}_q\|$
POMM	$(2n_1+n_2+3)\|\mathbb{Z}_q\|$	$-$

B. Performance Evaluation for AFEMM

In this section, we compare our scheme AFEMM with Jia's [41] and Zhang's [44], Li's [40], which are under the function model of $F(X^{(1)}, X^{(2)})$. In the following, we conduct the evaluation in terms of functions as well as communication, computation and storage overhead.

1) Functionality: Similarly, only our scheme AFEMM can verify the result in the efficient way by the client itself or by the third party, while Jia's [41] only verifies the result on its own and Zhang's [44] can only conduct the verification in public. Moreover, Jia's fails to protect data of output, even if the inputs uploaded by the client are encrypted. We can see the functions of our schemes is better according to the intuitionistic comparison in Table VII.

In function model of $F(X^{(1)}, X^{(2)})$, all the matrices of database $D^{(1)}$ and $D^{(2)}$ together with verification data will be sent to the server in advance. Therefore, though this operation is conducted for one time in amortized model, the communication overhead of this operation should not be ignored.

TABLE VII
COMPARISON OF FUNCTIONALITY FOR AFEMM

	Jia's [41]	Zhang's [44]	*AFEMM*
Self-Verification	Yes	No	Yes
Public Verification	No	Yes	Yes
Data Encryption	No	Yes	Yes

TABLE VIII
COMPARISON OF COMMUNICATION OVERHEAD FOR AEFMM

	Client to Server	
	Preprocessing	*Requesting*
Jia' [41]	$N_1(2n_1n_2\|\mathbb{Z}_q\|+n_1n_2\|\mathbb{G}\|+\|id\|)$ $+N_2(n_2n_3\|\mathbb{Z}_q\|+n_2n_3\|\mathbb{G}\|+\|id\|)$	$2N\|id\|$
Zhang' [44]	$N_1(n_1n_2\|\mathbb{Z}_q\|+n_1n_2\|\mathbb{G}\|+\|id\|)$ $+N_2(n_2n_3\|\mathbb{Z}_q\|+\|id\|)$	$2N\|id\|$
POMM	$N_1(n_1n_2\|\mathbb{Z}_q\|+3n_2\|\mathbb{G}\|+\|id\|)$ $+N_2(n_2n_3\|\mathbb{Z}_q\|+3n_2\|\mathbb{G}\|+\|id\|)$	$2N\|id\|$

2) Communication Overhead: In preprocessing phase, the communication overheads of sending the server related data of database $D^{(1)}$ are shown in Table VIII. Then, all the three schemes will send the computation request to the server, where the overhead is $2|id|$. While the overhead of receiving the results of three schemes is $n_1n_3|\mathbb{Z}_q|$, the client will all acquire verification proofs with communication overheads $|\mathbb{G}'|$, $n_1n_3|\mathbb{G}'|$, and $3n_3|\mathbb{G}|$ of schemes Jia's, Zhang's, and AFEMM, respectively.

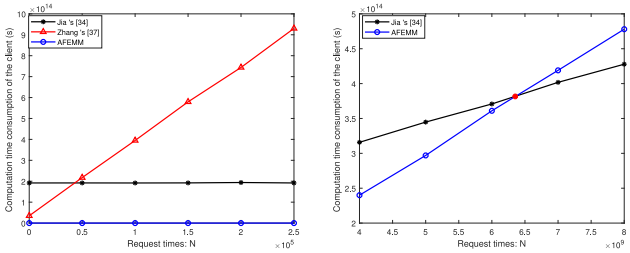
Although the cost of our scheme is higher than Jia's for N times request, the cost of verification tags of ours, i.e., $(N_1+N_2)3n_2|\mathbb{G}|$, is lower than other two schemes. To sum up, our scheme AFEMM is more efficient in communication overhead.

3) Computation Overhead: Comparing our scheme AFEMM with Jia's [41] and Zhang's [44] in computation overhead of client, we can obtain the Table IX, where there has four phases, i.e., Preprocessing, *Requesting*, *Verifying*, and *Decrypting*, involved. Because the client conducts main task of verification in [41], the verifying process must be pondered of this scheme.

In Preprocessing, we can see that for each matrix of database, Jia's conduct $O(n^3)$ operation in $\mathbb{Z}_q$, while the highest overhead of other two schemes is $O(n^2)$. As N_1 and N_2 is huge under this function model, our scheme is more efficient than Jia's and Zhang's according to the comparison in Table IX.

TABLE IX
COMPARISON OF COMPUTATION OVERHEAD FOR AFEMM

Phase	*Preprocessing*	*Requesting* (Multiple: N)	*Verifying* (Multiple: N)	*Decrypting* (Multiple: N)
Jia's [41]	$N_1[n_1n_2T_e + n_1n_2(2n_2+1)T_z]$ $+N_2[n_2n_3T_e + n_2n_3(2n_2+1)T_z] + 2T_i$	–	$n_1n_2T_z + T'$	–
Zhang's [44]	$N_1(2n_1n_2T_z + n_1n_2T_e) + N_2(n_2n_3T_z + n_3T_{e0})$	$n_1n_3T_z + n_1n_3T_e$	–	$(n_1n_2 + n_2n_3$ $+3n_1n_3 + n_2)T_z$
AFEMM	$N_1(6n_2T_e + 3n_2T_g + 3n_1n_2T_z)$ $+N_2(2n_2T_e + 5n_2T_g + 2n_2n_3T_z)$ $+2T_{e0} + (n_1+1)T_e + n_1n_2T_z$	$(4n_2+1)T_z + 3T_e + T_g$	–	$2n_1n_3T_z$



(a) (b)

Fig. 5. Comparison of computation time consumption of the client.

In online phases, the operation will be conducted for N-times. For Jia's scheme, the client takes huge time cost $n_1n_2T_z + T'$ in *Verifying*. In *Requesting* phase, the computation overhead of ours is $(4n_2+1)T_z + 3T_e + T_g$ which is lower than $n_1n_3T_z + n_1n_3T_e$ of Zhang's while in *Decrypting* computation overhead of ours is also smaller than Zhang's.

Taking all into account, though Jia's scheme is more efficient in online phases, it presents high cost in Preprocessing. Therefore, our scheme is optimal in three schemes.

Similarly, we conduct the experiment on the machine with 2.5GHz-processor and 4GB memory to prove above inference in reality. Utilizing time consumption to represent the computation overhead, we focus on the overhead influenced by he request times N.

Let all the matrices involved be the square matrices, where $n_1 = n_2 = n_3 = n = 10^5$. Considering the size of databases $D^{(1)}$ and $D^{(2)}$, we set $N_1 = N_2 = 10^4$. In Fig. 5(a), we can see that with the raise of N, the time consumption of Zhang's is always significantly higher than our scheme. The bigger N is, the higher the discrepancy is. As shown in Fig. 5(b), our scheme is more efficient than Jia's until N reach to almost 6.4×10^9. However, this value is closely related to the size of databases. With the huge databases, i.e., N_1 and N_2 are large, the value of N to reach the critical value can be ignored. In the conclusion, our scheme successfully present the advantage in computation.

4) Storage Overhead: According to the function model, the client will keep *id* of all the matrices in three schemes for requesting. Without decryption phase, Jia's [41] can only stored the secret key, the storage overhead of which is $(n_1+n_3)|\mathbb{Z}_q|$. As Zhang's [44] and AFEMM are should also keep the encryption parameters, the storage overhead of ours is $(4+2n_1)|\mathbb{Z}_q| + (N_1n_1 + N_2n_3)|\mathbb{Z}_q|$, which is lower than $|\mathbb{Z}_q| + N_1(n_1+n_2)|\mathbb{Z}_q| + N_2(n_2+n_3)|\mathbb{Z}_q|$, the overhead of Zhang's. Overall consideration, the storage overhead of our scheme AFEMM is acceptable.

VII. CONCLUSION

In this paper, on the basis of original scheme, we continually research on the verifiable outsourcing scheme for matrix multiplication over amortized model, which can be applied for two different functions of matrix multiplication particularly. We not only analyze the security of whole scheme, but also guarantee the correctness of results. Besides, our scheme outperforms other existing schemes in communication efficiency, computation ability, and storage overhead. In a word, this scheme presents improvements over old versions and has a much more promising prospect. Surely, as this paper only consider the operation of matrix multiplication, but the algorithms can also be applied to the operation of matrix-vector multiplication. In the future work, we can focus on the delegation of more linear operations.

REFERENCES

[1] M. Li *et al.*, "SPFM: Scalable and privacy-preserving friend matching in mobile cloud," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 583–591, Apr. 2017.

[2] Y. Li, L. Zhou, H. Zhu, and L. Sun, "Privacy-preserving location proof for securing large-scale database-driven cognitive radio networks," *IEEE Internet Things J.*, vol. 3, no. 4, pp. 563–571, Aug. 2016.

[3] H. Zhu *et al.*, "You can jam but you cannot hide: Defending against jamming attacks for geo-location database driven spectrum sharing," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 10, pp. 2723–2737, Oct. 2016.

[4] R. Hasan, M. Hossain, and R. Khan, "Aura: An IoT based cloud infrastructure for localized mobile computation outsourcing," in *Proc. Mobile Cloud Comput. Services Eng. (MobileCloud)*, vol. 17. San Francisco, CA, USA, 2015, pp. 183–188.

[5] H. Li, Y. Dai, L. Tian, and H. Yang, "Identity-based authentication for cloud computing," in *Proc. Cloud Comput.*, Beijing, China, 2009, pp. 157–166.

[6] J. Li, Y. Zhang, X. Chen, and Y. Xiang, "Secure attribute-based data sharing for resource-limited users in cloud computing," *Comput. Security*, vol. 72, pp. 1–12, Jan. 2018.

[7] P. Li *et al.*, "Privacy-preserving outsourced classification in cloud computing," *Clust. Comput.*, vol. 21, pp. 1–10, Apr. 2017.

[8] K. Fan, J. Wang, X. Wang, H. Li, and Y. Yang, "A secure and verifiable outsourced access control scheme in fog-cloud computing," *Sensors*, vol. 17, no. 7, 2017, Art. no. 1695.

[9] H. Li *et al.*, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Trans. Depend. Secure Comput.*, vol. 13, no. 3, pp. 312–325, May/Jun. 2016.

[10] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. Adv. Cryptol. (CRYPTO)*, Santa Barbara, CA, USA, 2010, pp. 465–482.

[11] J. Li, J. Li, X. Chen, C. Jia, and W. Lou, "Identity-based encryption with outsourced revocation in cloud computing," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 425–437, Feb. 2015.

[12] H. Li *et al.*, "EPPDR: An efficient privacy-preserving demand response scheme with adaptive key evolution in smart grid," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2053–2064, Aug. 2014.

[13] G. Brunette and R. Mogull, *Security Guidance for Critical Areas of Focus in Cloud Computing V2.1*, Cloud Security Alliance, 2009, pp. 1–76.

[14] H. Li, R. Lu, L. Zhou, B. Yang, and X. Shen, "An efficient merkle-tree-based authentication scheme for smart grid," *IEEE Syst. J.*, vol. 8, no. 2, pp. 655–663, Jun. 2014.

[15] G. Xu *et al.*, "Achieving efficient and privacy-preserving truth discovery in crowd sensing systems," *Comput. Security*, vol. 69, pp. 114–126, Aug. 2017.

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

[17] H. Li, D. Liu, Y. Dai, and T. H. Luan, "Engineering searchable encryption of mobile cloud networks: When QoE meets QoP," *IEEE Wireless Commun.*, vol. 22, no. 4, pp. 74–80, Aug. 2015.

[18] J. Li, Y. K. Li, X. Chen, P. P. C. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1206–1216, May 2015.

[19] H. Li, D. Liu, Y. Dai, T. H. Luan, and S. Yu, "Personalized search over encrypted data with efficient and secure updates in mobile clouds," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 1, pp. 97–109, Jan./Mar. 2017.

[20] C. Liu, C. Yang, X. Zhang, and J. Chen, "External integrity verification for outsourced big data in cloud and IoT: A big picture," *Future Gener. Comput. Syst.*, vol. 49, pp. 58–67, Aug. 2015.

[21] H. Y. Li, "Design of computer vision and image processing technology based car keys tooth identification detection system," in *Advanced Materials Research*, vol. 1079–1080. Pfäffikon, Switzerland: Trans Tech, 2015, pp. 1061–1063.

[22] W. Zhang, G. Zhang, Y. Wang, Z. Zhu, and T. Li, "NNB: An efficient nearest neighbor search method for hierarchical clustering on large datasets," in *Proc. IEEE Int. Conf. Semantic Comput. (ICSC)*, Anaheim, CA, USA, 2015, pp. 405–412.

[23] P. Li *et al.*, "Multi-key privacy-preserving deep learning in cloud computing," *Future Gener. Comput. Syst.*, vol. 74, pp. 76–85, Sep. 2017.

[24] H. Li *et al.*, "Achieving secure and efficient dynamic searchable symmetric encryption over medical cloud data," *IEEE Trans. Cloud Comput.*, to be published.

[25] S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations," in *Proc. Theory Cryptography Conf.*, Cambridge, MA, USA, 2005, pp. 264–282.

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

[27] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. STOC*, vol. 9. Bethesda, MD, USA, 2009, pp. 169–178.

[28] A. C. Yao, "Protocols for secure computations," in *Proc. IEEE 23rd Annu. Symp. Found. Comput. Sci.*, Chicago, IL, USA, 1982, pp. 160–164.

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

[30] M. Backes, D. Fiore, and R. M. Reischuk, "Verifiable delegation of computation on outsourced data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, Berlin, Germany, 2013, pp. 863–874.

[31] C. Gentry, "Toward basing fully homomorphic encryption on worst-case hardness," in *Proc. CRYPTO*, vol. 6223. Santa Barbara, CA, USA, 2010, pp. 116–137.

[32] K.-M. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Proc. Adv. Cryptol. (CRYPTO)*, Santa Barbara, CA, USA, 2010, pp. 483–501.

[33] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *Proc. TCC*, vol. 7194. Taormina, Italy, 2012, pp. 422–439.

[34] H. Li, D. Liu, Y. Dai, T. H. Luan, and X. S. Shen, "Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 1, pp. 127–138, Mar. 2015.

[35] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2389–2396, Sep. 2014.

[36] Y. Wu, K. Guo, J. Huang, and X. S. Shen, "Secrecy-based energy-efficient data offloading via dual connectivity over unlicensed spectrums," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3252–3270, Dec. 2016.

[37] X. Chen *et al.*, "New algorithms for secure outsourcing of large-scale systems of linear equations," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 1, pp. 69–78, Jan. 2015.

[38] C. Wang, K. Ren, and J. Wang, "Secure optimization computation outsourcing in cloud computing: A case study of linear programming," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 216–229, Jan. 2016.

[39] Z. L. Feng and R. Safavi-Naini, "Private outsourcing of polynomial evaluation and matrix multiplication using multilinear maps," in *Proc. Int. Conf. Cryptol. Netw. Security*, Paraty, Brazil, 2013, pp. 329–348.

[40] H. Li *et al.*, "Enabling efficient publicly verifiable outsourcing computation for matrix multiplication," in *Proc. IEEE Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Sydney, NSW, Australia, 2015, pp. 44–50.

[41] K. Jia, H. Li, D. Liu, and S. Yu, "Enabling efficient and secure outsourcing of large matrix multiplications," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, San Diego, CA, USA, 2015, pp. 1–6.

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

[43] J. Li, X. Huang, J. Li, X. Chen, and Y. Xiang, "Securely outsourcing attribute-based encryption with checkability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2201–2210, Aug. 2014.

[44] S. Zhang, H. Li, K. Jia, Y. Dai, and L. Zhao, "Efficient secure outsourcing computation of matrix multiplication in cloud computing," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Washington, DC, USA, 2016, pp. 1–6.

Authors' photographs and biographies not available at the time of publication.