JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 33, 635-652 (2017)
DOI: 10.6688/JISE.2017.33.3.3

Efficient Privacy-Preserving Building Blocks in Cloud Environments under Multiple Keys

HONG RONG, HUI-MEI WANG, JIAN LIU AND MING XIAN
State Key Lab of Complex Electromagnetic Environment Effects on Electronics and Information System
National University of Defense Technology
Changsha, 410073 P.R. China
E-mail: r.hong_nudt@hotmail.com; freshcdwhm@163.com;
ljabc730@gmail.com; qwertmingx@tom.com

With the rapid growth of big data, clients lack in storage and computational resources tend to outsource their data and computation tasks to cloud service providers. Due to concerns of privacy leakage in cloud, data owners encrypt the data via their own keys before outsourcing. However, it's rather difficult for cloud servers to perform computations over those encrypted data, since most of existing solutions are restricted to single key setting and may reveal the final output to adversary. In this paper, we propose two sets of privacy-preserving building blocks to support outsourced computation. Specifically, these schemes allow the cloud servers to evaluate basic arithmetic functions including addition, multiplication, and exponentiation over ciphertexts under different keys by utilizing the property of proxy re-encryption technique to transform ciphertexts. They are proven to be secure in the semi-honest model and secure against eavesdropping attacks. Experimental results on local testbed and real cloud environment demonstrate that the proposed solutions achieve great performance improvements with low overhead on data owners.

Keywords: big data, cloud computing, privacy-preserving data mining, building blocks, multiple keys

1. INTRODUCTION

Nowadays, the volume and details of data captured by organizations or companies are growing at tremendous speed, making it impossible for data owners with limited resources to store and process them locally. Since cloud computing technology offers highly scalable computational power in a cost-efficient way, outsourcing both data and computation tasks to cloud service providers becomes a natural solution. Despite enormous advantages the cloud offers, security issues are impeding clients from utilizing those advantages [1]. In order to protect data privacy, the data require to be encrypted before outsourcing. Therefore, to perform arbitrary computation tasks over the encrypted data regardless of the underlying encryption scheme has become a hot research topic.

Another important evolution of outsourced computation is that the datasets stored in the cloud may come from multiple data owners. Computing over the joint datasets can significantly enhance the value of big data, which may be applied to create new services and derive information that cannot be acquired from individual datasets [2]. For instance, combining medical records from a large number of patients can improve the accuracy of disease diagnosis. Naturally, different data owners use different keys for encryption to preserve their sensitive data, yet, making computation outsourcing more formidable, as

Received June 29, 2016; revised September 28, 2016, accepted October 22, 2016.
Communicated by Ram Chakka.

most of existing approaches only work in the situation where the uploaded data are encrypted under the same key [3-5]. A straightforward solution is to transform the ciphertexts under multiple keys into encryptions under a unified key. Then, privacy-preserving protocols are easily to be designed as the single-key setting. BCP cryptosystem with double decryption property can be used to unify encryption keys [6], since the master secret key is able to decrypt any ciphertexts under any user's key. Whereas the major drawback is that the leakage of the master key poses great threat to the entire system. Secure multi-party computation [7, 8] also allows cooperative computation on data contributed by multiple owners without disclosing their own inputs, but this kind of scheme relies on interactions between users instead of a powerful third party, and leaks the final output to all participants. In short, current solutions for outsourcing computation under multiple keys are not secure and efficient enough in practical applications.

In this paper, to solve the aforementioned problems, we propose two sets of schemes for Outsourced Privacy-preserving Building Blocks (OPBB), namely, $OPBB^{+}$ and $OPBB^{*}$. These schemes are constructed based on Proxy Re-Encryption (PRE) technique, which allows a user to obtain the result of arbitrary outsourced function over the encrypted data under multiple keys without compromising privacy of its input, output, and the federate datasets. Specifically, the main contributions of the paper are three-fold:

- Firstly, our proposed building blocks allow to evaluate basic arithmetic functions, including addition, multiplication, as well as exponentiation. Not only the inputs and function parameters are encrypted under different keys, but also the output is protected in encrypted form. With OPBB, cloud users are able to compute most of secure computations, like scalar product, high-degree polynomial, *etc.* Apart from this, the schemes can be adopted in privacy-preserving data mining algorithms.
- Secondly, OPBB consists of two schemes, *i.e.*, $OPBB^{+}$ and $OPBB^{*}$, based on additively and multiplicatively homomorphic property of the underlying cryptosystems, respectively. In $OPBB^{+}$, we present a Secure Multiplication (SM^{+}) protocol which supports multiplication over ciphertexts from different parties, and a Secure Exponentiation (SE^{+}) protocol which enables the cloud to compute exponent with encrypted power and base. Likewise, $OPBB^{*}$ is composed of Secure Addition protocol (SA^{*}) and Secure Exponentiation (SE^{*}) protocol. They allow the cloud servers to conduct corresponding calculations in a privacy-preserving manner without leaking additional information to each of them.
- Thirdly, theoretical analysis shows that the proposed building blocks are secure in the semi-honest model and resistant against eavesdropping. The computational complexity is much less than similar works with minimum client involvements. Extensive experiments on local testbed and Amazon cloud show that our OPBB solutions work efficiently and scale well in both computation and communications.

The rest of the paper is organized as follows. Our system model and threat model are presented in Section 2. In Section 3, we briefly introduce proxy re-encryption techniques. We describe the design details of our privacy-preserving building blocks under multiple keys in Section 4, followed by security analysis in Section 5. Performance evaluations are given in Section 6. In Section 7, we discuss some related work. Finally, we summarize the paper and outline future work in Section 8.

2. PROBLEM STATEMENT

In this section, we formally describe our system model, threat model and design goals.

2.1 System Model

There are n data owners $\{O_1, \ldots, O_n\}$ in our system model, who hold their own data denoted by $\{D_1, \ldots, D_n\}$ along with their respective public/private key pairs, denoted by $\{pk_1/sk_1, \ldots, pk_n/sk_n\}$. As depicted in Fig. 1, O_i uploads its encrypted data $\mathsf{Enc}_{pk_i}(D_i)$ under pk_i to cloud for storage, where $i \in [1, n]$. There's also a cloud user U who submits query q to the cloud for computing a function $f(D_1, \ldots, D_n, q)$ over the federate database. Let pk_σ/sk_σ denote U's key pair and $\mathsf{Enc}_{pk_\sigma}(q)$ denote the encryption of q. We adopt well-known two-server model [9, 10], denoted by C_1 and C_2. C_1 provides massive data storage and interacts with O_i and U directly to process their queries, while C_2 holds its key pair pk_u/sk_u and assists C_1 to achieve complex computation. Besides, we assume both servers have tremendous computation power.

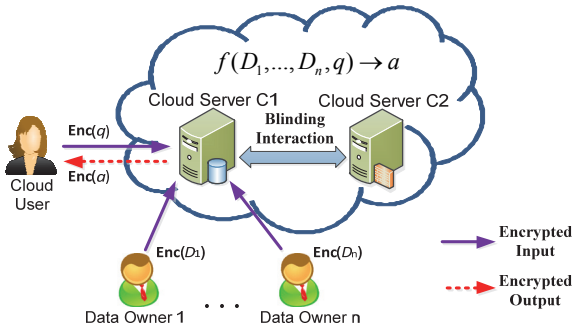

Fig. 1. System model.

The process for computation in cloud environment can be illustrated as follows. Initially, cloud servers and data owners generate their respective public and private key pairs based on public system parameters. Next, C_1 produces the re-encryption key for each O_i through interactions with O_i and C_2. After $\mathsf{Enc}_{pk_i}(D_i)$ are uploaded to the cloud, C_1 transforms these ciphertexts into encryptions under sk_u by re-encryption technique. When U submits $\mathsf{Enc}_{pk_\sigma}(q)$ to C_1, servers begin to evaluate f with the encrypted dataset $\{\mathsf{Enc}_{pk_1}(D_1), \ldots, \mathsf{Enc}_{pk_n}(D_n), \mathsf{Enc}_{pk_i\sigma}(q)\}$ through a set of privacy-preserving protocols which will be discussed in Section 4. In the end, U is able to retrieve the encrypted result by using its own secret key.

The function f should stand for most arithmetic computations, for instance, a general multivariate polynomial whose inputs are jointly provided by multiple parties like the following function:

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} c_i x_i^{d_i}, \tag{1}$$

where the coefficients $\{c_i | i \in [1, n]\}$, the degrees $\{d_i | i \in [1, n]\}$, and data $\{x_i | i \in [1, n]\}$ are

belonged to different owners, and they are encrypted under different keys before outsourcing. Note that this is a stronger security requirement than work [11, 12], since the coefficients and degrees are known to public in their work. Based on Eq. (1), our privacy-preserving building blocks must address three basic operations over encrypted data, including addition, multiplication, as well as exponentiation.

2.2 Threat Model

According to the security definitions by Goldreich [13], there are two common threat models: semi-honest and malicious. A semi-honest adversary follows the protocol, but may keep a record of all interactions and use it to infer additional information, while a malicious adversary may behave arbitrarily to breach data privacy. In this paper, we assume all parties in our system model, including cloud servers, data owners, and cloud users, are semi-honest. This assumption is reasonable in real world, since the reputation of public cloud providers may be severely ruined, once deviations from the protocol are detected. Security against a semi-honest adversary $\mathcal{A}$ can be proven by giving inputs to a simulator built in the ideal world. If the output of this simulator is computationally distinguishable from the real protocol execution, then the protocol is secure in the semi-honest model. We also assume the adversary is capable to perform eavesdropping over the network and analyze the intercepted traffic so long as $\mathcal{A}$ compromises one of participants. Moreover, collusion attacks are not considered in this paper.

2.3 Design Goals

Given the model above, our proposed schemes should satisfy requirements on correctness, confidentiality, and efficiency as follows:

- **Correctness:** If cloud servers and all users follow the designed protocols, the final response should be decrypted to the correct result of the target function.
- **Confidentiality:** During the outsourced process, nothing private regarding the content of each user's inputs, intermediate results, or the outputs should be revealed to cloud servers, or other parties.
- **Efficiency:** The cloud servers should take charge of most outsourced computation and process queries with relatively small latency, meanwhile the workload and communication overhead on clients' side should be minimized.

3. PRELIMINARIES

Proxy Re-Encryption (PRE) is a practical scheme proposed by Blaze *et al.* [14]. In a PRE system, a proxy is given a re-encryption key $rk_{i\to j}$ so that it can transform a ciphertext under public key pk_i into an encryption of the same plaintext under another user's public key pk_j. The proxy, however, learns nothing regarding the corresponding plaintext.

PRE schemes can be categorized into two types: bidirectional and unidirectional PRE. If the re-encryption key $rk_{i\leftrightarrow j}$ allows the proxy to convert ciphertext under pk_i into ciphertexts under pk_j and vice versa, then the scheme is bidirectional. If $rk_{i\to j}$ merely al-

lows the proxy to convert from pk_i to pk_j, then the scheme is unidirectional. In this work, we use the classic bidirectional PRE scheme based on ElGamal cryptosystem [15]. It consists of the following five algorithms:

- $\mathsf{KeyGen}(\mathbb{G}, p, g) \rightarrow \{pk_i, sk_i\}$: Let $\mathbb{G}$ be a multiplicative cyclic group of an order of p, and g be a generator of $\mathbb{G}$. U_i uses this key generation algorithm to generate a key pair $sk_i = a \in \mathbb{Z}_{p^*}$ and $pk_i = g^a \in \mathbb{G}$.
- $\mathsf{ReKeyGen}\{sk_i, sk_j\} \rightarrow \{rk_{i \leftrightarrow j}\}$: The re-encryption key generation algorithm takes two private keys sk_i and sk_j as inputs, and outputs a re-encryption key $rk_{i \leftrightarrow j} = sk_j/sk_i$. Here, it is required that $i \neq j$ in that there's no point to re-encrypt own ciphertext.
- $\mathsf{Enc}(pk_i, m) \rightarrow \{CT_i\}$: The encryption algorithm takes a public key pk_i and a message $m \in \mathcal{M}$ as inputs. It outputs a ciphertext $CT_i = (mg^r, pk_i^r)$ under pk_i. Here, $\mathcal{M}$ denotes the message space, and r is a random number in $\mathbb{Z}_{p^*}$.
- $\mathsf{ReEnc}\{rk_{i \leftrightarrow j}, CT_i\} \rightarrow \{CT_j\}$: The re-encryption algorithm takes a re-encryption key $rk_{i \leftrightarrow j}$ and an original ciphertext CT_i as inputs, and outputs a transformed ciphertext $CT_j = (m \cdot g^r, (pk_i^r)^{rk_{i \leftrightarrow j}})$ under pk_j.
- $\mathsf{Dec}(sk_i, CT_i) \rightarrow \{m\}$: The decryption algorithm takes a private key sk_i and an original or converted ciphertext CT_i under pk_i. It outputs a plaintext message $m = m \cdot g^r / ((pk_i^r)^{1/sk_i})$.

Furthermore, ElGamal encryption has multiplicatively homomorphic property over ciphertexts. More specifically, we have the following:

$$\mathsf{Enc}_{pk}(m_1) \times \mathsf{Enc}_{pk}(m_2) = \mathsf{Enc}_{pk}(m_1 \cdot m_2) \tag{2}$$

$$\mathsf{Enc}_{pk}(m_1)^{\alpha} = \mathsf{Enc}_{pk}(m_1^{\alpha}) \tag{3}$$

where $m_1, m_2 \in \mathcal{M}$, and "$\cdot$" denotes the multiplication operation in the plaintext domain, while "$\times$" denotes the multiplication operation in the ciphertext domain.

4. PROPOSED PRIACY-PRESERVING BUILDING BLOCKS

Our solution for generic outsourced computation has two diverse secure building blocks, that is, OPBB$^+$ and OPBB*. Specifically, OPBB$^+$ is constructed based on additive homomorphism of variant PRE while OPBB* is constructed based on original PRE. In each scheme, we propose a set of protocols addressing such operations that cannot be evaluated by C_1 independently via homomorphic property. In the following sections, we first present the scheme for key generation and distribution. Next, we describe the design details of OPBB$^+$ and OPBB* respectively.

4.1 Key Distribution Protocol

In the beginning, the key management authority initializes the cryptosystem parameters composed of a multiplicative cyclic group $\mathbb{G}$ with order p, and group generator g. Then, these parameters are distributed to all parties, including cloud servers as well as clients. Each party generates its key pairs by running $\mathsf{KeyGen}(\mathbb{G}, p, g)$, including O_i's key (pk_i, sk_i) for $1 \leq i \leq n$, U's key (pk_σ, sk_σ) and C_2's unified key (pk_u, sk_u). After that,

servers and data owners generate re-encryption keys in a collaborative manner. First, C_1 produces a random number r_i in group $\mathbb{Z}_{p^*}$ (Hereafter, we use the notion $r \in_R \mathbb{Z}_{p^*}$ to denote random number generation), and sends it to O_i who computes r_i/sk_i. Then, O_i delivers r_i/sk_i to C_2. C_2 later blinds the message by multiplying it by sk_u and sends it to C_1. In the end, C_1 is able to gain the re-encryption key between O_i and C_2 via $(r_i sk_u/sk_i) \cdot r_i^{-1}$. The overall communication is protected by secure communication protocol like SSL. It's evident that sk_i is unknown to cloud servers, since the intermediate result is blinded by r_i. sk_u is unknown to C_1, because the final input is randomized by sk_i. Moreover, we assume all the public keys are accessible to C_1.

4.2 Construction of OPBB$^+$

OPBB$^+$ is constructed based on a variant version of ElGamal encryption. Given that $m \in \mathcal{M}$, the encrypted form is like $CT = (g^m \cdot g^r, g^{ar})$ instead of $CT = (m \cdot g^r, g^{ar})$ mentioned in Section 3. This change contributes to additively homomorphic property, which can be described as below:

$$\mathsf{Enc}_{pk}(m_1) \times \mathsf{Enc}_{pk}(m_2) = \mathsf{Enc}_{pk}(m_1 + m_2) \tag{4}$$

$$\mathsf{Enc}_{pk}(m_1)^{\alpha} = \mathsf{Enc}_{pk}(m_1 \cdot \alpha) \tag{5}$$

The decryption requires solving a discrete logarithm, but no efficient general method for computing discrete logarithms on conventional computers has been proposed. However, a large lookup table or Pollard's Kangaroo algorithm [16] can be used to solve this efficiently as long as the message size is small enough.

OPBB$^+$ mainly consists of two sub-protocols, secure addition protocol and secure exponentiation protocol.

A. Secure Multiplication (SM$^+$) Protocol:

Given that C_1 holds private inputs $\mathsf{Enc}_{pk_u}(a)$, $\mathsf{Enc}_{pk_u}(b)$, and C_2 holds the private key sk_u, the goal of this protocol is compute the encryption of multiplication of a and b, *i.e.*, $\mathsf{Enc}_{pk_u}(a \cdot b)$. This protocol cannot be conducted by C_1 alone, by contrast, requiring close interactions between C_1 and C_2. This indicates that decryption is an indispensable part of execution. Since it is a computational intensive operation in virtue of solving discrete logarithm, our design tries to reduce the count of decryption operations to its minimum. The steps of SM$^+$ protocol are shown as follows:

Step 1: C_1 generates two random numbers $r_1, r_2 \in_R \mathbb{Z}_{p^*}$, and computes $X_1 = \mathsf{Enc}_{pk_u}(a)^{r_1}$, $X_2 = \mathsf{Enc}_{pk_u}(b)^{r_2}$. Then, X_1, X_2 are sent to C_2.

Step 2: C_2 decrypts X_2 by running $h = \mathsf{Dec}(sk_u, X_2)$ with its private key sk_u, and computes $Y = X_1^h$. Afterwards, Y is sent to C_1. Obviously, it can be seen that $h = r_2 \cdot b$ and $Y = \mathsf{Enc}_{pk_u}(r_1 \cdot r_2 \cdot a \cdot b)$ based on Eq. (5).

Step 3: Upon receiving Y, C_1 computes $Y^{(r_1 r_2)^{-1}}$ to obtain the encrypted $a \cdot b$. The correctness can be easily proved in the following:

$$\begin{aligned} Y^{(r_1 r_2)^{-1}} &= \mathsf{Enc}_{pk_u}(r_1 \cdot r_2 \cdot a \cdot b)^{r_1^{-1} r_2^{-1}} \\ &= \mathsf{Enc}_{pk_u}(r_1 \cdot r_2 \cdot a \cdot b \cdot r_1^{-1} \cdot r_2^{-1}) \\ &= \mathsf{Enc}_{pk_u}(ab) \end{aligned} \tag{6}$$

Besides, note that a, b, and random numbers cannot be arbitrarily large in that the decryption has restriction on the plaintext size.

B. Secure Exponentiation (SE^+) Protocol:

Given that C_1 holds private input $\mathsf{Enc}_{pk_u}(a)$, $\mathsf{Enc}_{pk_u}(b)$, and C_2 holds the private key sk_u, the goal of this protocol is compute the encryption of exponentiation of b with base a, *i.e.*, $\mathsf{Enc}_{pk_u}(a^b)$. However, it is rather formidable to design such a secure protocol that does not disclose any privacy under two-server setting, because the underlying scheme cannot be used to randomize plaintext with an exponent. Thus, we adopt the third party P which is also semi-honest but non-colluding with others to assist the computation. P who can be any cloud user holds its public/private key pair pk_P/sk_P and pk_u. The details of SE^+ protocol are shown as follows:

Step 1: C_1 generates four random numbers r_1, r_2, r_3, $r_4 \in_R \mathbb{Z}_{p^*}$, and computes the encryptions $X_1 = \mathsf{Enc}_{pk_u}(a)^{r_1}$, $X_2 = \mathsf{Enc}_{pk_u}(b) \times \mathsf{Enc}_{pk_u}(r_2)$, $X_3 = \mathsf{Enc}_{pk_u}(a)^{r_3}$, $X_4 = \mathsf{Enc}_{pk_u}(b)^{-1} \times \mathsf{Enc}_{pk_u}(r_4)$. Moreover, C_1 performs $H_1 = \mathsf{ReEnc}(rk_{u\to P}, X_3)$, $H_2 = \mathsf{ReEnc}(rk_{u\to P}, X_4)$ with re-encryption key $rk_{u\to P}$, $R_1 = \mathsf{Enc}_{pk_P}(r_1)$, and $R_2 = \mathsf{Enc}_{pk_P}(r_2)$. Then, X_1, X_2 are sent to C_2, while H_1, H_2, R_1, R_2 are sent to P.

Step 2-1: C_2 decrypts X_1, X_2, and gets their plaintexts denoted by Y_1, Y_2, respectively. Later, C_2 computes $K_1 = \mathsf{Enc}_{pk_u}(Y_1^{Y_2})$ and sends it to C. It can be observed that $K_1 = \mathsf{Enc}_{pk_u}((r_1 a)^{b+r_2})$ according to Eqs. (4) and (5).

Step 2-2: P decrypts H_1, H_2 with sk_P, and obtains the plaintexts denoted by h_1, h_2, respectively. By decrypting R_1, R_2, P recovers r_1, r_2. Then, P computes $K_2 = \mathsf{Enc}_{pk_u}(h_1^{-r_2})$, $K_3 = \mathsf{Enc}_{pk_u}(r_1^{h_2})$. It can be seen that $K_2 = \mathsf{Enc}_{pk_u}((r_3 a)^{-r_2})$, and $K_3 = \mathsf{Enc}_{pk_u}(r_1^{r_4 - b})$. These two ciphertexts K_2, K_3 are transmitted to C_1.

Step 3: Once receiving K_1, K_2, K_3 from other parties, C_1 computes $S_1 = \mathsf{Enc}_{pk_P}(r_1^{-r_2})$, $S_2 = \mathsf{Enc}_{pk_P}(r_1^{-r_4})$, $S_3 = \mathsf{Enc}_{pk_P}(r_3^{r_2})$. It is able to calculate the target encryption $\mathsf{Enc}_{pk_u}(a^b)$ by multiplying those encryptions with SM^+ protocol. The correctness can be verified as follows:

$$\begin{aligned} \mathrm{SM}^+(K_1, K_2, K_3, S_1, S_2, S_3) &= \mathsf{Enc}_{pk_u}(Y_1^{Y_2} \cdot h_1^{-r_2} \cdot r_1^{h_2} \cdot r_1^{-r_2} \cdot r_1^{-r_4} \cdot r_3^{r_2}) \\ &= \mathsf{Enc}_{pk_u}((r_1 a)^{r_2 + b} \cdot (r_3 a)^{-r_2} \cdot r_1^{r_4 - b - r_2 - r_4} \cdot r_3^{r_2})) \\ &= \mathsf{Enc}_{pk_u}(a^b). \end{aligned} \tag{7}$$

4.3 Construction of OPBB*

Differing from the above solution, OPBB* is constructed based on ElGamal scheme with the multiplicatively homomorphic property. In other words, C_1 can evaluate multi-

plication over encrypted data without interactions with C_2. By contrast, addition of ciphertexts still requires interactions, but there is no restriction on the length of plaintext.

A. Secure Addition (SA*) Protocol:

Assume that C_1 holds private inputs $\mathsf{Enc}_{pk_u}(a)$, $\mathsf{Enc}_{pk_u}(b)$, and C_2 holds the secret key sk_u. The goal of this protocol is to compute the encrypted addition of a and b, *i.e.*, $\mathsf{Enc}_{pk_u}(a+b)$ as output to C_1. As the encryption system is only partial homomorphic for multiplication, we adopt blinding approach like SM+. Nevertheless, it has been proven that the two non-colluding servers setting may disclose private information about inputs ratio [17]. A straightforward approach is to send $\mathsf{Enc}_{pk_u}(ra)$, $\mathsf{Enc}_{pk_u}(rb)$ to C_2, but the blinding factor r can be simply removed by computing $ra/rb \rightarrow a/b$, where the randomized ra and rb are known to C_2. Their security enhanced solution needs three non-colluding servers to complete. However, in this paper, we still consider two-server model while guaranteeing that there's no privacy leakage about inputs and outputs.

In this protocol, we leverage ElGamal blinding ciphertext technique, denoted as $\mathsf{Blind}(C, r)$. It is an operation that randomizes c_1 of ciphertext C by multiplying random value r so that the plaintext b is blinded by r, where $C = (c_1, c_2)$, $c_1 = rbg^{r'}$, $r, b \in \mathbb{G}$, and $r' \in_R \mathbb{Z}_{p^*}$. This operation only requires one multiplication over $\mathbb{G}$. The overall steps of SA* are presented as follows:

Step 1: C_1 generates random numbers: $r_1, r_2 \in_R \mathbb{Z}, r_3, r_4 \in_R \mathbb{G}$, and ensure that r_1, r_2 holds the relation that $r_1 + r_2 \equiv 2 \bmod N$, where N is used to generate multiplicative cyclic group $\mathbb{G}$. C_1 then computes $L_1 = \mathsf{Enc}_{pk_u}(a) \times \mathsf{Enc}_{pk_u}(b)$, $L_2 = \mathsf{Blind}(\mathsf{Enc}_{pk_u}(a)^2, r_3)$, $L_3 = \mathsf{Blind}(L_1, r_1 \cdot r_3)$, $L_4 = \mathsf{Blind}(\mathsf{Enc}_{pk_u}(b)^2, r_4)$, $L_5 = \mathsf{Blind}(L_1, r_2 \cdot r_4)$. Finally, C_1 sends two pairs $\{L_2, L_3\}$, $\{L_4, L_5\}$ to C_2. It can be verified that $L_2 = \mathsf{Enc}_{pk_u}(a \cdot b)$, $L_2 = \mathsf{Enc}_{pk_u}(r_3 \cdot a^2)$, $L_3 = \mathsf{Enc}_{pk_u}(r_1 r_3 ab)$, $L_4 = \mathsf{Enc}_{pk_u}(r_4 \cdot b^2)$, $L_5 = \mathsf{Enc}_{pk_u}(r_2 \cdot r_4 \cdot a \cdot b)$.

Step 2: C_2 decrypts each element by running $L_i' = \mathsf{Dec}(sk_u, L[i])$, for $i \in [2, 5]$, and computes $S_1 = L_2' + L_3'$ and $S_2 = L_4' + L_5'$. After that, $\{S_1, S_2\}$ is encrypted as $\{S_1', S_2'\}$, which are sent back to C_1.

Step 3: After receiving $\{S_1', S_2'\}$, C_1 generates random number $r_5 \in \mathbb{G}$, meanwhile it computes $\alpha_1 = \mathsf{Blind}(S_1', r_3^{-1} \cdot r_5)$, $\alpha_2 = \mathsf{Blind}(S_2', r_4^{-1} \cdot r_5)$. It can be inferred that $\alpha_1 = \mathsf{Enc}_{pk_u}(r_5 a^2 + r_1 r_5 \cdot a \cdot b)$, $\alpha_2 = \mathsf{Enc}_{pk_u}(r_5 b^2 + r_2 r_5 \cdot a \cdot b)$. α_1 and α_2 are sent to C_2.

Step 4: C_2 decrypts $\{\alpha_1, \alpha_2\}$, and gains the corresponding plaintexts, denoted by α_1', α_2' respectively. Then, C_2 computes the summation $\lambda = \alpha_1' + \alpha_2'$ and encrypts it as λ' using pk_u. Ultimately, λ' is transmitted to C_1.

Step 5: Upon receiving λ', C_1 is able to obtain the desired encryption of $a+b$ by blinding with r_5^{-1}. The former result is powered by $2^{-1} \bmod p$. The correctness of this protocol can be verified in the following equation:

$$\begin{aligned}\mathsf{Blind}(\lambda', r_5^{-1})^{2^{-1}} &= \mathsf{Enc}_{pk_u}((r_5 a^2 + r_1 r_5 ab + r_5 b^2 + r_2 r_5 ab) \cdot r_5^{-1})^{2^{-1}} \\ &= \mathsf{Enc}_{pk_u}((a^2 + b^2 + (r_1 + r_2)ab)^{2^{-1}})\end{aligned}$$

$$= \mathsf{Enc}_{pk_u}(((a+b)^2)^{2^{-1}})$$
$$= \mathsf{Enc}_{pk_u}(a+b). \quad (8)$$

B. Secure Exponentiation (SE*) Protocol:

Given that C_1 holds private input $\mathsf{Enc}_{pk_u}(a)$, $\mathsf{Enc}_{pk_u}(b)$, and C_2 holds the private key sk_u, the goal of this protocol is compute the encryption of exponentiation of b with base a, *i.e.*, $\mathsf{Enc}_{pk_u}(a^b)$. Thanks to the multiplicatively homomorphic property of underlying encryption scheme, exponentiation of the ciphertext has the equal effect on its corresponding plaintext, as shown by Eq. (3). This makes it possible to blind the exponent with random values whereas SE$^+$ cannot do similar work without third party. The complete steps are presented as follows:

Step 1: C_1 generates three random numbers $r_1 \in_R \mathbb{G}$, r_2, $r_3 \in_R \mathbb{Z}_{p^*}$, and computes $X_0 = \mathsf{Blind}(\mathsf{Enc}_{pk_u}(a), r_1)$, $X_1 = X_0^{r_3}$, $X_2 = \mathsf{Blind}(\mathsf{Enc}_{pk_u}(b), r_2)$, as well as $X_3 = \mathsf{Enc}_{pk_u}(r_1)$. Then, X_1, X_2, X_3 are sent to C_2. Based on Eqs. (1) and (2), we can infer that $X_1 = \mathsf{Enc}(a \cdot r_1)^{r_3})$, $X_2 = \mathsf{Enc}_{pk_u}(b \cdot r_2)$.

Step 2: C_2 decrypts X_1, X_2 by running $Y_1 = \mathsf{Dec}(sk_u, X_1)$, $Y_2 = \mathsf{Dec}(sk_u, X_2)$. It also recovers r_1 from X_3 with sk_u. After that, C_2 computes $Z_1 = \mathsf{Enc}_{pk_u}(Y_1^{Y_2})$, $Z_2 = \mathsf{Enc}_{pk_u}(r_1^{Y_2})$, which are then sent to C_1.

Step 3: After receiving Z_1, Z_2, C_1 calculates two exponentiations: $H_1 = Z_1^{r'_2 r'_3}$, and $H_2 = Z_2^{r'_2}$, where $r'_2 = r_2^{-1} \bmod p$ and $r'_3 = r_3^{-1} \bmod p$. The final encryption of a^b can be achieved by $H_1 \times H_2$. Its correctness proof is given below:

$$\begin{aligned} H_1 \times H_2 &= \mathsf{Enc}_{pk_u}((Y_1^{Y_2})^{r_2^{-1} r_3^{-1}}) \times \mathsf{Enc}_{pk_u}((r_1^{Y_2})^{-r_2^{-1}}) \\ &= \mathsf{Enc}_{pk_u}((a \cdot r_1)^{r_3 \cdot b \cdot r_2 \cdot r_2^{-1} \cdot r_3^{-1}} \cdot r_1^{-b \cdot r_2 \cdot r_2^{-1}}) \\ &= \mathsf{Enc}_{pk_u}(a^b \cdot r_1^b \cdot r_1^{-b}) \\ &= \mathsf{Enc}_{pk_u}(a^b). \end{aligned} \quad (9)$$

5. SECURITY ANALYSIS

In this section, we analyze the security of the proposed building blocks, focusing on how our solutions can preserve the privacy of each data owners' dataset and the result from analyzing and eavesdropping attacks by the adversarial servers and data owners.

5.1 Analysis of OPBB$^+$

Due to Composition Theorem, we evaluate the security of each protocol in OPBB$^+$ by using "Real-vs.-Ideal" framework in the semi-honest model [13]. Note that the security of our protocol is based on the well-known "blinding technique", which means that the original plaintext is randomized with a random value by using Eq. (5).

(1) Security of SM^+: For C_1 corrupted by adversary $\mathcal{A}_1$, its view in the real world includes $\{\mathsf{Enc}_{pk_u}(a), \mathsf{Enc}_{pk_u}(b), \mathsf{Enc}_{pk_u}(ab), X_1, X_2, Y\}$, which are encrypted under pk_u. We can build a simulator in the ideal world to simulate the view of $\mathcal{A}_1$ by encryption of random numbers. Due to the semantic security of the ElGamal cryptosystem, $\mathcal{A}_1$ is computationally infeasible to distinguish the real world from the ideal world. For C_2 corrupted by $\mathcal{A}_2$, its real world view includes $\{X_1, X_2, Y\}$, and $\{r_1a, r_2a, r_1r_2ab\}$. We can build a simulator in the ideal world by executing the protocol with random numbers as inputs. Since the inputs a, b are blinded by randomly distributed numbers r_1 and r_2, $\mathcal{A}_2$ cannot distinguish from the real world and the ideal world.

(2) Security of SE^+: For C_1 corrupted by $\mathcal{A}_1$, its real world view comprises six encryptions $\{\mathsf{Enc}_{pk_u}(a), \mathsf{Enc}_{pk_u}(b), \mathsf{Enc}_{pk_u}(a^b), K_1, K_2, K_3\}$. We build a simulator in the ideal world that produces six ciphertexts of random values. $\mathcal{A}_1$ is not able to distinguish two worlds because C_1 has no knowledge about the private key sk_u. For C_2 compromised by $\mathcal{A}_2$, the real world view includes $\{X_1, X_2, Y_1, Y_2, K_1\}$. Y_1, Y_2 are the decryptions of X_1, X_2, respectively, but they are randomized by blinding factor r_1, r_2. Thus, it is computationally hard for $\mathcal{A}_2$ to distinguish the real world from the ideal world. As for P corrupted by $\mathcal{A}_3$, the view of real world includes inputs $\{H_1, H_2, R_1, R_2\}$, the corresponding plaintexts $\{h_1, h_2, r_1, r_2\}$, as well as outputs $\{K_1, K_2\}$. Though $\mathcal{A}_3$ can perform decryption with sk_P, the sensitive data a, b are still blinded by random numbers r_3, r_4, and they cannot be removed through exponential operation. Therefore, $\mathcal{A}_3$ is not able to distinguish the real world from the ideal world created by our simulator.

(3) Security against Eavesdropping Attacks: Supposing that an adversary $\mathcal{A}_e$ has compromised one party in our system model, such as O_i, C_1, or C_2, $\mathcal{A}_e$ may launch eavesdropping by using the corrupted party's private key. During the data uploading stage, $\mathcal{A}_e$ corrupting O_i may capture O_j's data ($i \neq j$), but he cannot decrypt them since $\mathcal{A}_e$ does not possess O_j's private key. During outsourcing stage, $\mathcal{A}_e$ corrupting data owners also cannot decrypt the traffic captured between the servers, because they're encrypted under pk_u. As for the corrupted C_1, no privacy of intercepted data are revealed to $\mathcal{A}_e$ who only knows the re-encryption keys. Moreover, $\mathcal{A}_e$ corrupting C_2 also knows nothing about original data or final results, since they're either blinded by random values, or encrypted under the client's own private key. Note that two or more adversaries compromising multiple parties are not considered in this paper.

5.2 Analysis of OPBB*

Similar to the security analysis of the first scheme, we prove that the OPBB* is secure under semi-honest model by using "Real-vs.-Ideal" framework [13].

(1) Security of SA^*: For C_1 corrupted by adversary $\mathcal{A}_1$, its view in the real world includes inputs $\{\mathsf{Enc}_{pk_u}(a), \mathsf{Enc}_{pk_u}(b), S_1', S_2', \lambda'\}$, final outputs $\{\mathsf{Enc}_{pk_u}(a+b)\}$ and intermediate results $\{L_1, L_2, L_3, L_4, \alpha_1, \alpha_2\}$. We build a simulator in the ideal world that generates the same number of encrypted random values. Since $\mathcal{A}_1$ does not know the private key, if $\mathcal{A}_1$ could discern the real world and ideal world, it indicates that $\mathcal{A}_1$ could have an algorithm to distinguish ciphertexts generated by ElGamal, which contradicts to the assumption that the encryption scheme is semantically secure. Thereby, it's computationally difficult for $\mathcal{A}_1$ to discern the two views. For C_2 corrupted by $\mathcal{A}_2$, the

real world view can obtain the plaintexts r_3a^2, r_1r_3ab, r_4b^2, r_2r_4ab, $r_5b^2 + r_1r_5ab$, $r_5b^2 + r_2r_5ab$ by using its secret key. Obviously, $\mathcal{A}_2$ cannot directly obtain either the plaintexts of a and b or their ratio like a/b, because they are blinded by random numbers r_1, r_2, r_3, r_3, r_5. Though $\mathcal{A}_2$ may know that $r_1 + r_2 \equiv 2 \bmod N$, it's still not enough to set up equations because the relation $r_1 = 2 - r_2$ does not necessarily hold. We build a simulator in the ideal world to simulate the view of $\mathcal{A}_2$ by using random values as inputs and executing the steps in the protocol. If the blinding numbers are randomly distributed, it's infeasible for $\mathcal{A}_2$ to make differentiation between the real world and the ideal world.

(2) Security of SE*: For C_1 corrupted by $\mathcal{A}_1$, the view of real world consists of inputs $\{\mathsf{Enc}_{pk_u}(a), \mathsf{Enc}_{pk_u}(b), Z_1, Z_2\}$, outputs $\{X_1, X_2, X_3, \mathsf{Enc}_{pk_u}(a^b)\}$. Recall that all the inputs and outputs are ciphertexts encrypted under pk_u. We build a simulator in the ideal world by performing the same computation with random values. Thus, due to the security of the encryption scheme, $\mathcal{A}_1$ is not able to distinguish from the real world and the ideal world without sk_u. For C_2 compromised by $\mathcal{A}_2$, its real world view includes inputs $\{X_1, X_2, X_3\}$, outputs $\{Z_1, Z_2\}$. Meanwhile, $\mathcal{A}_2$ also gets the plaintexts $\{Y_1, Y_2, r_1\}$ by decryption. Then, we build a simulator in the ideal world that conducts the same algorithm but using random values as inputs. Nevertheless, Y_1 is randomized by r_1, r_3 while Y_2 is blinded by r_2, because $Y_1 = (a \cdot r_1)^{r_3}$, $Y_2 = b \cdot r_2$. As long as the hard problem of discrete logarithm holds, $\mathcal{A}_2$ cannot compute the base of Y_1 even if he has r_1. So $\mathcal{A}_2$ cannot distinguish the real world from the ideal world.

(3) Security against Eavesdropping Attacks: OPBB* scheme is also secure against eavesdropping attacks on account of different keys used by different parties for encryption. The proof part can be referred to OPBB+.

6. PERFORMANCE ANALYSIS

In this section, we analyze the performance of our two schemes from both theoretical and experimental perspectives, and compare our work with similar methods.

6.1 Theoretical Analysis

Let Exp, Mul, and Dlog denote modular exponentiation, multiplication, discrete logarithm, respectively. In our comparison, we mainly focus on the similar schemes which enable outsourced computation under multiple keys. For data owners and cloud users, their costs merely restricted to encryption, decryption, and limited participations.

A. Complexity of OPBB+

Table 1 shows the comparison of complexity between our building block and similar works. The addition costs for all these schemes are 2Mul, because they are additively homomorphic. SM+ requires 9Exp+2Mul+1Dlog computation of cloud servers and $6|\mathbb{G}|$ bits for interactive communication. It can be seen that the similar schemes Vitamin+ [17] and PTK [18] consume larger computation overhead for secure multiplication. Note that since solving Dlog depends on the message size, our scheme cannot be compared with PTK directly. However, it takes PTK 10Exp + 18Mul computation to re-encrypt one ciphertext whereas it only takes 1Exp for the other two schemes. Besides, their commu-

nication costs are the same. As for outsourced exponentiation, it costs cloud servers 95Exp+27Mul+11Dlog to complete SE^+ with 18|𝔾| traffic load, while there're no implementations on this kind of computation in both works [17, 18].

Table 1. The complexity comparison of additively homomorphic schemes.

Algorithm Cost	$OPBB^+$	$Vitamin^+$	PTK
Addition	2Mul	2Mul	2Mul
Multiplication	9Exp+2Mul+1Dlog	11Exp+5Mul+2Dlog	20Exp+32Mul
Exponentiation	95Exp+27Mul+11Dlog	–	–

Table 2. The complexity comparison of multiplicatively homomorphic schemes.

Algorithm Cost	$OPBB^*$	$Vitamin^*$
Addition	15Exp+20Mul	24Exp+32Mul
Multiplication	2Mul	2Mul
Exponentiation	16Exp+13Mul	–
Server Count	2	3

B. Complexity of $OPBB^*$

In SA^* protocol, it costs servers 15Exp + 20Mul and 18|𝔾| bits communication overhead. As shown in Table 2, compared to the similar method $Vitamin^*$ [17] with 24Exp + 32Mul computation and 18|𝔾| communication cost, our scheme requires less computations and no third server. For secure exponentiation outsourcing, SE^* requires 16Exp + 13Mul computation and 10|𝔾| bits overhead for cloud servers. Obviously, it works faster than secure addition protocol, because the homomorphic property can be used to blind the exponent.

6.2 Experimental Analysis

We evaluate the computation cost and communication overhead of the proposed schemes by performing experiments on local testbed and real cloud environment. All protocols are implemented in C++ using Crypto++5.6.3 library. As for algorithms which require solving discrete logarithm, we generate a large lookup table and adopt binary search algorithm. To achieve practical security level, we choose $|p| = 1024$ bits for ElGamal encryption, and $|N| = 1024$ bits for BCP encryption in PTK [18].

A. Experiments on Local Testbed

The configurations of our local servers are Intel Xeon E5-2620@2.10 GHz CPU with 8 GB RAM running CentOS 6.5 with 1Gbps bandwidth. To test the performance of our schemes for secure addition and multiplication, we choose $f(\vec{x}, \vec{y}) = \sum_{i=1}^{m} x_i \cdot y_i$ as the outsourcing function, where $\vec{x}, \vec{y}$ are two m-dimension vectors. In the test, each element is a randomly generated integer with 32-bit length, that is, $x_i, y_i \in [1, 2^{32} - 1]$, $i \in [1, m]$. The two vectors are encrypted under separate keys before outsourcing. We measure the computation time and communication traffic during the outsourcing period.

First, we assess the time that cloud clients spend on encrypting their vectors. The results are presented in Fig. 2. It can be easily observed that the computation time for each client grows with the increase of vector dimension size, while PTK scheme takes

much more time than the other schemes. The difference in time mainly arises from the fact that BCP cryptosystem utilized by PTK scheme as the underlying encryption scheme incurs more complexity than ElGamal adopted by the rest building blocks because of twice larger modulus size. For m = 1000, OPBB$^+$ and OPBB* require each data owner about 1.77s to encrypt their vector, which is relatively small compared with cloud overhead.

Fig. 3 shows the re-encryption time for current schemes grows linearly with the increase of dimension size. However, the time of PTK scheme rises more sharply than the others. The reason is that the PRE scheme allows re-encryption to be completed by proxy server independently while PTK requires heavy interactions between two servers.

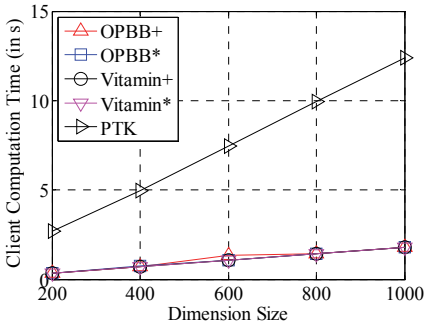

Fig. 2. Client overhead vs. vector dimension.

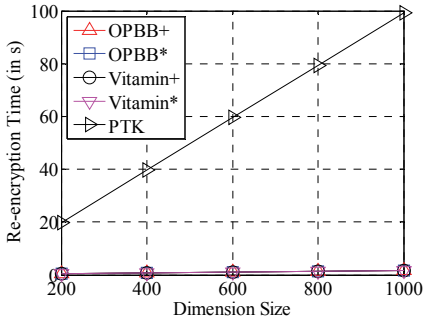

Fig. 3. Re-encryption time vs. vector dimension.

Next, we evaluate the computation time and communication overhead for cloud servers. The computation process for cloud servers consists of re-encryption, as well as scalar product computation while the communication overhead is brought by two-server interactions. As shown in Figs. 4 and 5, we can see that both the computation and communication overhead at cloud side increase with dimension size. Fig. 4 reveals that for additively homomorphic methods, our OPBB$^+$ saves 37.06%, 97.46% computation time compared with Vitamin$^+$ and PTK, respectively. As for multiplicatively homomorphic methods, the processing time of OPBB* is only half as much as that of Vitamin*. Note that the re-encryption cost of PTK accounts for almost half of its entire outsourcing time. The results are also consistent with theoretical analysis in previous section.

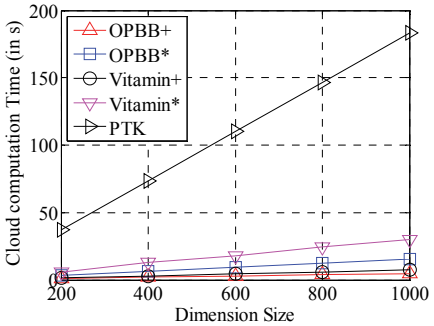

Fig. 4. Cloud computation vs. dimension.

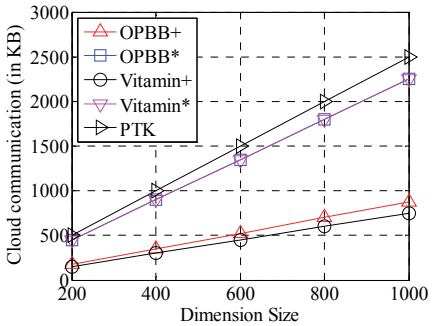

Fig. 5. Cloud communication vs. dimension.

From Fig. 5, we can observe that the communication cost of PTK remains the largest compared with other schemes, while multiplicatively homomorphic schemes consume 2.5 times traffic as much as additively homomorphic schemes, since OPBB* and Vitamin* require one more communication round than OPBB$^+$ and Vitamin$^+$. Moreover, OPBB$^+$ seems more efficiently than OPBB*, because searching for an index within a small lookup table is faster than expensive exponential operations. This suggests that the cloud can leverage OPBB$^+$ to accelerate computation performance for short messages.

B. Experiments on Amazon EC2

In order to analyze the performance on real cloud environment, we conduct tests on two Amazon EC2 On-Demand instances [19], the type of which is m4.large with 2vCPU and 8GB memory running Red Hat Enterprise Linux. The function f is a high-degree polynomial which integrates addition, multiplication, and exponentiation operations. Specifically, $f(x) = \sum_{i=1}^{n} c_i x^{d_i}$, in which the coefficient c_i and the degree d_i are contributed by O_i, while x is provided by U requesting cloud servers to compute $f(x)$. Let n be the number of data owners. All the data randomly generated on the condition that c_i, $x \in [1, 10^8]$ and $d_i \in [1, 10^3]$. Remark that the congregated data are encrypted under their owners' public keys. Since there's no existing work that assumes every part of polynomial is encrypted, we only implement the outsourcing protocol based on our proposed building blocks.

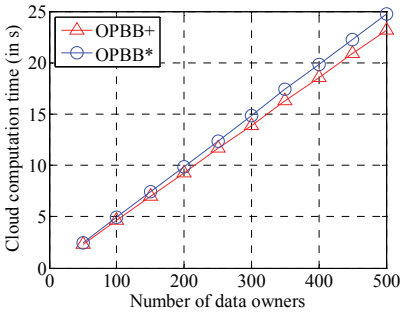

Fig. 6. Cloud computation vs. owner counts.

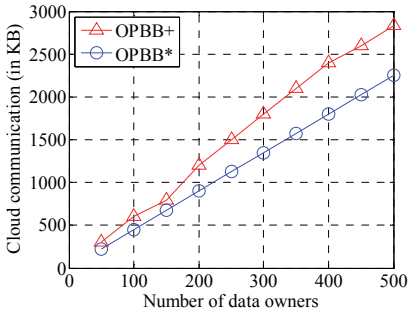

Fig. 7. Cloud communication vs. owner counts.

We evaluate the computation and communication overhead on cloud servers. Figs. 6 and 7 show the performance of our methods scale linearly with the number of data owners. However, it's apparent to find that OPBB* outperform OPBB$^+$ this time, for there's a lot of exponentiation operations in $f(x)$ and the sub-protocol SE$^+$ incurs much more computational and communication cost than SE*. For instance, when n = 500, it takes OPBB* 23.15s to compute $f(x)$ with 2.2MB while OPBB$^+$ needs 24.74s and 2.76MB to complete the same work. The results indicate that OPBB* is a kind of better option for outsourced computation involved with exponentiation.

7. RELATED WORK

In order to protect data privacy from access by unauthorized adversary, data are

usually encrypted before outsourcing. However, how to perform arbitrary computations over the encrypted data remains an open question. Fully Homomorphic Encryption (FHE), first proposed by Gentry [20], allows both addition and multiplication operation over the ciphertexts. Though a lot of new schemes regarding FHE have been proposed to improve security and efficiency, the complexity of those cryptosystems are still too high to be used in real application [21]. To enhance performance, some works designed a set of privacy-preserving primitives based on additive homomorphic property of Paillier cryptosystem, which are used in outsourced *k*-Nearest Neighbor classification [3-5]. However, these schemes incur heavy computation and communication overhead.

To compute a function securely over the database distributed among multiple parties, Secure Multi-party Computation (SMC) [7] was introduced to address this issue. But this technique is not fit for computation outsourcing, since SMC based approach assumes data are not encrypted and held by each participant, and the final result is generally exposed to public, whereas both the datasets and result are encrypted in our system model. Besides, SMC protocols exert heavy workloads on clients on contrary to our assumption that they have weak computation power [24, 25]. Recently, Jung *et al.* [11, 12] proposed several protocols which enable multiple parties to evaluate multivariate polynomial jointly by aggregating their data without secure channel. This system model is similar with ours. However, their product and sum protocols reveal the final output to all participants and the aggregator. In addition, the coefficients and powers are not encrypted as we do.

Most of aforementioned solutions are based on assumption that a single encryption key is shared among data owners, but they cannot be used to perform calculation over ciphertexts under multi-keys. To address this issue, BCP cryptosystem with double trapdoor decryption mechanism was utilized to transform ciphertexts under different keys into encryptions under a single public key by using master secret key [18]. Then, cryptographic protocols were proposed to evaluate arithmetic circuits. However, the major drawback of their schemes is that the master secret key has the power to decrypt all kinds of ciphertexts, which makes the server who holds the key a highly valuable target. Besides, ciphertext transformation requires complex interactions between the servers. To improve efficiency, Wang *et al.* [17] proposed two schemes Vitamin* and Vitamin$^+$ by leveraging proxy re-encryption with partially homomorphic properties. Both schemes support the basic operations (*i.e.*, additions and multiplications), but they are not secure and efficient enough in that Vitamin* may directly reveal the ratio of inputs without a third cloud server. Our schemes are also constructed on PRE; nevertheless, they're proven to be secure with only two servers under the semi-honest model, and they support exponentiation apart from addition and multiplication. Another relevant work [22, 23] utilized random matrix transformation to preserve security in collaborative outsourced data mining. Although this method is much more efficient than public key encryption, it cannot be used as privacy-preserving building blocks to compute generic computations. Our previous work [26] leveraged encryptions of random numbers to prevent leaking input ratio with high probability during secure addition process, but it's not as secure as method proposed in this paper.

8. CONCLUSION

In this paper, we proposed two sets of privacy-preserving building blocks for out-

sourcing computation under two-server model, called OPBB^{+} and OPBB* for short. By leveraging PRE transformation, these schemes allow cloud servers to collaboratively calculate a generic function with the data encrypted under owners' respective keys. Additionally, only the legitimate query user can retrieve the final result, which further protects user privacy. Theoretical analysis shows that the proposed building blocks can evaluate addition, multiplication, and exponentiation without revealing the privacy of the data and results in the semi-honest model. Experiments on outsourced scalar product and polynomial functions have also demonstrated efficiency of our solution. As future work, we plan to investigate more secure and efficient solutions with the stringent threat model that the adversaries may have some background knowledge of the outsourced database and are able to perform collusion attacks.

REFERENCES

1. K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, Vol. 16, 2012, pp. 69-73.
2. S. Eubank, H. Guclu, G. S. A. Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang, "Modeling disease outbreaks in realistic urban social networks," *Nature*, Vol. 429, 2004, pp. 180-184.
3. Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure *k*-nearest neighbor query over encrypted data in outsourced environments," in *Proceedings of the 30th IEEE International Conference on Data Engineering*, 2014, pp. 664-675.
4. B. K. Samanthula, Y. Elmehdwi, and W. Jiang, "*K*-nearest neighbor classification over semantically secure encrypted relational data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 27, 2015, pp. 1-14.
5. F. Li, R. Shin, and V. Paxson, "Exploring privacy preservation in outsourced K-nearest neighbors with multiple data owners," in *Proceedings of the 7th ACM Cloud Computing Security Workshop*, 2015, pp. 53-64.
6. E. Bresson, D. Catalano, and D. Pointcheval, "A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications," in *Proceedings of ASIACRYPT*, 2003, pp. 37-54.
7. S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," *IACR Cryptology ePrint Archive*, Vol, 2011, 2011, pp. 435-451.
8. H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud trhough privacy homomorphism,", in *Proceedings of the 27th IEEE International Conference on Data Engineering*, 2011, pp. 217-290.
9. S. S. M. Chow, J. H. Lee, and M. Strauss, "Two-party computation model for privacy-preserving queries over distributed databases," in *Proceedings of the 16th Annual Network and Distributed System Security Conference*, 2009, pp. 1-16.
10. M. D. Van and A. Juels, "On the impossibility of cryptography alone for privacy-preserving cloud computing," in *Proceedings of the 5th USENIX Workshop on Hot Topics in Security*, 2010, pp. 1-8.
11. T. Jung, X. Mao, X. Li, S. Tang, W. Gong, and L. Zhang, "Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation," in *Pro-*

ceedings of the 32nd Annual IEEE International Conference on Computer Communications, 2013, pp. 2634-2642.
12. T. Jung, X. Li, and M. Wan, "Collusion-tolerable privacy-preserving sum and product calculation without secure channel," *IEEE Transactions on Dependable and Secure Computing*, Vol. 12, 2015, pp. 45-57.
13. O. Goldreich, *The Foundations of Cryptography*, Cambridge University Press, Vol. 2, Ch. Basic Applications, 2004.
14. M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proceedings of EUROCRYPT*, 1998, pp. 127-144.
15. T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, Vol. 31, 1985, pp. 469-472.
16. J. M. Pollard, "Monte Carlo methods for index computation mod *p*," *Mathematics of Computation*, Vol. 32, 1978, pp. 918-924.
17. B. Wang, M. Li, S. S. M. Chow, and H. Li, "Computing encrypted cloud data efficiently under multiple keys," in *Proceedings of the 4th International Workshop on Security and Privacy in Cloud Computing*, 2013, pp. 504-513.
18. A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IEEE Transactions on Information Forensics and Security*, Vol. 8, 2013, pp. 2046-2058.
19. Amazon EC2 Pricing, https://aws.amazon.com/ec2/pricing/on-demand/.
20. C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st ACM Symposium on Theory of Computing*, 2009, pp. 169-178.
21. Z. Brakerski, C. Gentry, S. Halevi, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory*, Vol. 18, 2011, pp. 169-178.
22. Y. Huang, Q. Lu, and Y. Xiong, "Collaboraive outsourced data mining for secure cloud computing," *Journal of Networks*, Vol. 9, 2014, pp. 2655-2664.
23. Q. Lu, Y. Xiong, X. Gong, and W. Huang, "Secure collaborative data mining with multi-owner in cloud computing," in *Proceedings of the 11th IEEE International Conference on Trust*, *Security and Privacy in Computing and Communications*, 2012, pp. 100-108.
24. R. Lu, X. Lin, and X. She, "Spoc: A secure and privacy-preserving opportunistic computing framework for mobile-healthcare emergency," *IEEE Transactions on Paralell and Distributed Systems*, Vol. 24, 2013, pp. 614-624.
25. J. Zhou, Z. Cao, X. Dong, and X. Lin, "PPDM: A privacy-preserving protocol for cloud-assited e-healthcare systems," *IEEE Journal of Selected Topics in Signal Processing*, Vol. 9, 2015, pp. 1332-1344.
26. H. Rong, H. Wang, K. Huang, J. Liu, and M. Xian, "Privacy-preserving scalar product computation in cloud envrionments under multiple keys," in *Proceedings of the 17th International Conference on Intelligent Data Engineering and Automated Learning*, 2016, pp. 248-258.

Hong Rong received the M.S. degree in School of Electronic Science and Technology from National University of Defense Technology in 2013. He is currently working toward the Ph.D. degree in the State Key Laboratory of CEMEE, National University of Defense Technology. His research interests include privacy-preserving data mining, cloud computing security.

Hui-Mei Wang received the B.S. degree from Southwest Jiaotong University in 2004, M.S. degree and Ph.D. degree from National University of Defense Technology in 2007 and 2012. She is currently a Lecturer in the State Key Laboratory of CEMEE at National University of Defense Technology. Her research interests include network security, cloud computing and distributed systems.

Jian Liu is a Lecturer in the CEMEE State Key Laboratory at National University of Defense Technology. He received the B.S., M.S., and Ph.D. degrees from National University of Defense Technology in 2009, 2011, and 2016 respectively. His research interests include secure storage and computation outsourcing in cloud computing and security of distributed systems.

Ming Xian received the B.S. degree (1991), M.S. degree (1995), and Ph.D. degree (1998) from National University of Defense Technology. Now he is a Professor in the State Key Laboratory of CEMEE at National University of Defense Technology. His research interests include on cryptography and information security, cloud computing, wireless sensor network and system modeling, simulation and evaluation.