Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 413265, 7 pages
http://dx.doi.org/10.1155/2014/413265

Research Article

Two-Cloud-Servers-Assisted Secure Outsourcing Multiparty Computation

Yi Sun,[1] Qiaoyan Wen,[1] Yudong Zhang,[2] Hua Zhang,[1] Zhengping Jin,[1] and Wenmin Li[1]

[1] *State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2] *Brain Image Processing, Columbia University, New York, NY 10032, USA*

Correspondence should be addressed to Yi Sun; sybupt@bupt.edu.cn

Received 19 February 2014; Accepted 13 April 2014; Published 28 May 2014

Academic Editors: A. Miné and B. Sun

Copyright © 2014 Yi Sun et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We focus on how to securely outsource computation task to the cloud and propose a secure outsourcing multiparty computation protocol on lattice-based encrypted data in two-cloud-servers scenario. Our main idea is to transform the outsourced data respectively encrypted by different users' public keys to the ones that are encrypted by the same two private keys of the two assisted servers so that it is feasible to operate on the transformed ciphertexts to compute an encrypted result following the function to be computed. In order to keep the privacy of the result, the two servers cooperatively produce a custom-made result for each user that is authorized to get the result so that all authorized users can recover the desired result while other unauthorized ones including the two servers cannot. Compared with previous research, our protocol is completely noninteractive between any users, and both of the computation and the communication complexities of each user in our solution are independent of the computing function.

1. Introduction

Secure multiparty computation (SMC) [1–7] is dedicated to computing a certain function among a set of mutually distrusted participants on their private inputs without revealing private information. Informally speaking, assuming that there are m participants, $P_1, P_2, \ldots, P_m$, each of them has a private number, respectively, $x_1, x_2, \ldots, x_m$. They want to cooperate to compute the function $y = f(x_1, x_2, \ldots, x_m)$ without revealing x_i of P_i to other parties $P_j, j \neq i, i, j \in \{1, \ldots, m\}$, as well as guaranteeing that any unauthorized ones cannot get the result y. In the past, researchers mainly focused on designing the style of secure multiparty computation protocols by which users themselves cooperatively accomplish the function evaluation through their internal interactions [1, 3, 8–11]. The computation and communication complexities always depend polynomially on the complexity of the function to be computed. Therefore, users suffer from the heavy overload of these protocols.

The emergence of the cloud [12, 13] inspires users to apply the powerful computing ability of the cloud to help them to conduct complicated computations, that is, secure outsourcing computation [14–18] to the cloud. They expect that the cloud can independently complete any function computation on their outsourced data although the data has been encrypted by their own keys for security. Moreover, the final result should be kept private to the cloud even though it is the cloud that conducts all of the computations about the computing function. In this way, users only need to encrypt their data and decrypt the returned message to get the desired result. All computations about the computing function are in the charge of the cloud. There are no interactions between any users, and the computation and communication complexities of each user are independent of the computing function. However, this expectation is proven to be impossible in the single cloud server setting due to the impossibility of program obfuscation [19]. Therefore, in this paper, we try to realize it by introducing one more cloud server to the original model described above. More precisely, we consider the following scenario.

There are $m + 2$ distrusted parties including m users and two cloud servers in our system. We assume that all of them act semihonestly. The m users $P_1, P_2, \ldots, P_m$, with each having a private input, respectively, x_i, as well as a pair of

public-private keys (pk_i, sk_i), $i = 1, \ldots, m$, encrypt their respective private inputs by their own public keys and then upload the ciphertexts of the inputs to a cloud server. They want to obtain the value $y = f(x_1, x_2, \ldots, x_m)$ even if they may not be aware of what the computing function $f(\cdot)$ is by applying two cloud servers to operate on the outsourced encrypted data without revealing $x_1, x_2, \ldots, x_m$ and the result y.

In this paper, we study the outsourcing computation problem in multiple users-two-cloud-servers scenario and propose a two-cloud-servers-assisted secure outsourcing multiparty computation protocol to compute any function on lattice-based encrypted data under multiple keys of the users. Herein, we apply one cloud server called the storing-cloud (SC) to store the outsourced data encrypted by users and make a midtransformation to these ciphertexts once some function begins to be computed. We call the other cloud server the computing-cloud (CC). It is responsible for transforming the midtransformed ciphertexts by SC to the ones that are blinded by the same two private keys of the two assisted cloud servers so that CC can further compute y following the function on the ciphertexts. Finally, in order to protect the result, the two servers cooperatively produce a custom-made result for each user. Compared with previous solutions, our protocol has the following three advantages.

(1) Our protocol is completely noninteractive between any users.

(2) The cloud is to do all of the computations related to the computing function, while users would do nothing except for encrypting their private inputs and decrypting the returned result.

(3) The computation and communication complexities of each user in our solution are independent of the computing function.

Organization. The rest of this paper is organized as follows. In Section 2, we briefly give an overview of some recent related works. Herein, we consider the problems in secure outsourcing computation from the point of view of the users and the cloud servers, respectively, and then rationally construct our protocol in the multiple users-two-cloud-servers setting. In Section 3, we briefly introduce a lattice-based encryption scheme and the security model and then present our protocol in Section 4 in detail. In Section 5, we analyze the proposed protocol in detail and give a strict proof based on real-ideal simulation paradigm. Finally, we summarize our work of this paper in the last section.

2. Related Works

According to previous research, there are many problems to be considered when outsourcing private data for function computation to the cloud. We discuss the difficulties in secure outsourcing computation to the cloud from the following two aspects.

(1) To Users: Privacy of the Inputs and Results. In secure outsourcing computation, users have to contribute their private data as the inputs of the function while not participating in the computation process. Moreover, all parties of the protocol including all users and cloud servers are mutually distrusted. Therefore, users would not like to submit their private data to the cloud. Allowing for security, a usual solution is to encrypt the private data before outsourcing them to the cloud. And there are some basic encryption models according to the encryption keys that users used.

In 2009, Gentry [20] presented a model where all users use a joint public key to encrypt their own private inputs while sharing the private key. Therein, the cloud cannot obtain the inputs or the result because they are protected by the encryption scheme, while the cloud does not have the private key. However, users have to participate in another interactive protocol to firstly recover the private key and then achieve the desired result. The processes, producing a joint public key, sharing the private key, and jointly recovering the result by their shared private key, bring large number of additional interactions among users, which is contrary to our expectation that we want to design a secure protocol with the least communications. Encrypting private data by the joint public key is not so satisfactory either. In cloud outsourcing scenario, it means that there are no interactions among the users whatsoever and the least two rounds of inevitable interactions between the user and the cloud server, sending out the inputs and receiving the result. Therefore, we look forward to a protocol with the least communications as well as low computations and high security. A recent work by Asharov et al. [21] proposes a scheme where users utilize their own public keys to encrypt their inputs, respectively, and guarantee that the cloud can succeed in computing the function on their private inputs by computing on the ciphertexts of the inputs encrypted under different keys. Although users still have to interact to obtain the result in the last step, encrypting respective input by the public key of each user is the best encryption model so far.

As to the privacy of the result, in 2011, Halevi et al. [22] proposed a noninteractive protocol to securely realize outsourcing computation. Therein, the server is entitled to learn the result. However, the computing result may be the vital information to the users in some scenario and so it cannot be revealed to others. Hence, besides the security of the inputs discussed above, users must consider the security of the result when constructing protocols. It should guarantee that any unauthorized users are not able to get the result although they may contribute their inputs and the cloud servers are not able to get the result although the result is computed by them. To this aspect, [20] has already protected the result by a joint public key of the users. However, this method is still not satisfactory since each authorized user is also not able to get it individually.

(2) To Cloud Servers: Feasibility of Operating on Encrypted Inputs. As discussed above, users would like to upload the encrypted inputs under their respective public keys to the cloud server rather than the original inputs. Therefore, the cloud servers, whose task is to compute a function on users' private inputs, would only obtain the ciphertexts of the inputs. That means that the cloud has to compute the function

(i) KeyGen(1^k): sample a ring element vector $a \leftarrow R_q^n$ and a ring element s from the distribution χ, denoted as $s \leftarrow \chi$, a ring element vector x from the distribution χ^n, denoted as $x \leftarrow \chi^n$. Then, the private key is $sk = s$; the public key is $p = as + 2x \in R_q^n$.
(ii) Enc(pk, m): sample $e \leftarrow \chi^n$ and compute $c_0 := \langle p, e \rangle + m \in R_q$ and $c_1 := \langle a, e \rangle \in R_q$. Output the ciphertext $c := (c_0, c_1) \in R_q^2$.
(iii) Dec(sk, c): compute $\mu = c_0 - c_1 s \in R_q$ and output $m' := \mu(\mathrm{mod}2)$.

Algorithm 1

on users' private inputs through performing corresponding computations on the ciphertexts of the inputs encrypted by different public keys of users. As we know, fully homomorphic encryption (FHE) [20, 23] can operate on the ciphertexts of the inputs to compute the desired result produced by the inputs. But the usual FHE schemes are single-key schemes in the sense that they only can perform computations on ciphertexts encrypted under the same key. It is not feasible to conduct computations on the ciphertexts encrypted under different keys. In order to solve this problem, López-Alt et al. [24] propose a new FHE called multikey fully homomorphic encryption (MFHE) which has applied the techniques of bootstrapping, modulus reduction, and relinearization to operate on the ciphertexts of the inputs encrypted by multiple, unrelated keys. When outsourcing private data to the cloud, user can firstly encrypt it by its own key by applying MFHE. It is indeed the optimal solution from the point of view of the feasibility of ciphertexts and the privacy of inputs. However, as we mentioned before, it is still not satisfactory because users need to evaluate the decryption key and then use it to recover the result interactively by participating in another SMC protocol.

In fact, according to [19], it is proved that it is indeed impossible to construct a completely noninteractive protocol in the single server setting due to the impossibility of program obfuscation. Hence, if we want to obtain a secure protocol with complete noninteraction of users in outsourcing computation, we need at least two cloud servers.

In brief, allowing for the privacy of inputs and results from the perspective of users as well as the feasibility of operating on the outsourced encrypted data from the perspective of the cloud servers, if we want to construct a completely noninteractive secure outsourcing multiparty computation protocol where the computation and communication complexities of each user are independent of the computing function, we have the following conclusions.

(1) All private data should be encrypted by the owners themselves using their respective public keys before outsourcing to the cloud servers.

(2) The returned messages for each user should be different so that all authorized users can recover the final result by their respective private key but the unauthorized ones cannot.

(3) It is reasonable to consider it in two-cloud-servers scenario.

3. Preliminaries

3.1. Lattice-Based Encryption. Since the privacy of the inputs and the computation complexity of each user depend on the encryption algorithm that the user used, an encryption scheme outstanding in both security and efficiency is the right one that users want to adopt. Hence, lattice-based encryption, which is against quantum attacks and is much more efficient than RSA and even the elliptic curve cryptosystem, becomes the first choice of rational users. Herein, we will show how the two cloud servers deal with the outsourced data encrypted by the lattice-based public key encryption scheme proposed in [24, 25] (denoted as LE scheme in this paper). Specifically, we recall it as follows.

Notations. Let k be the security parameter. Then, the LE scheme is parameterized by a prime $q = q(k)$, a degree n polynomial $f(x) \in \mathbb{Z}[x]$, and an error distribution χ over the ring $R_q = \mathbb{Z}_q[x]/\langle f(x) \rangle$. The parameters n, f, q, and χ are public. It assumes that, given the security parameter k, there are polynomial-time algorithms that output f and q and a sample from the error distribution χ.

The LE encryption scheme consists of the following three algorithms: KeyGen($\cdot$), Enc($\cdot$), and Dec($\cdot$) (Algorithm 1).

In [24], they apply the techniques of bootstrapping, modulus reduction, and relinearization to realize and to operate on the ciphertexts of the inputs encrypted by multiple, unrelated keys. Therein, they have obtained a secure outsourcing multiparty computation protocol on lattice-based encrypted data under multiple keys of users in one server scenario. However, it is not satisfactory because the interaction in the decryption stage is still inevitable.

In this paper, based on this encryption scheme, we consider the outsourcing problem in two-cloud-servers scenario and succeed to construct a secure noninteractive outsourcing protocol that achieves the least computation and communication complexities for users.

3.2. Security Model. In this paper, we will discuss our protocol in the semihonest model and analyze its security using the real-ideal paradigm [5].

Firstly, in the ideal world, the computation of the functionality $\mathcal{F}$ on users' private inputs is conducted by an additional trusted party that receives x_i from user P_i, $i = 1, 2, \ldots, m$, and returns the result $f(x_1, x_2, \ldots, x_m)$ to the authorized users P_t, while other unauthorized parties do not get any output. Hence, in the ideal world, all users' private

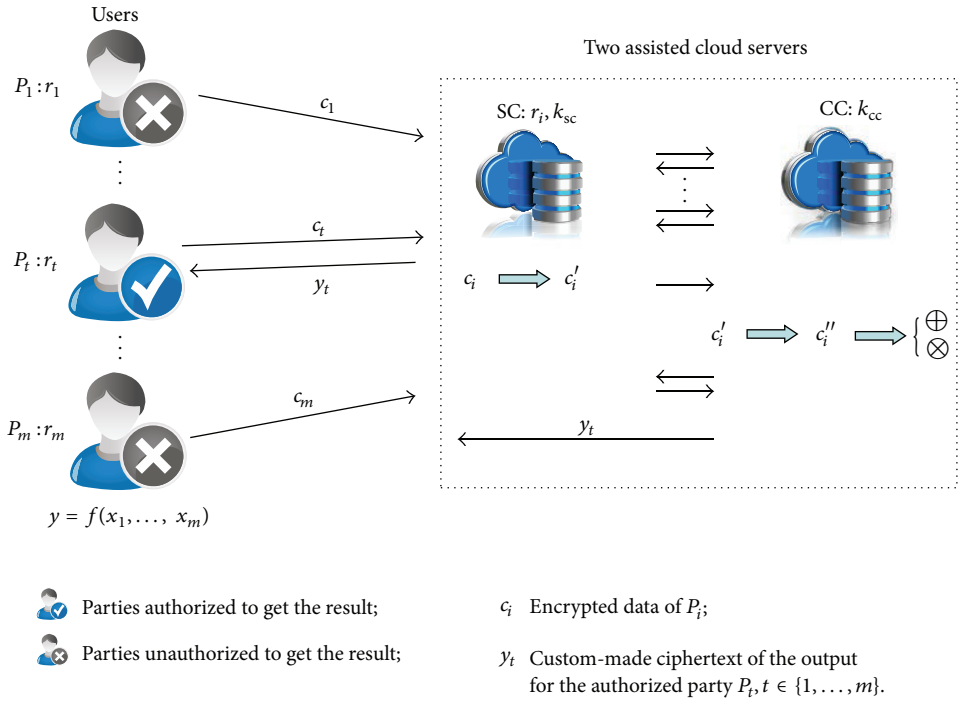

Figure 1: Framework of our construction.

inputs are well protected, and only authorized users are able to learn about the result. However, there is no trusted party in the real world, and so all parties have to run a protocol Π to get the desired result. During executing the protocol Π, all parties act semihonestly following the protocol but make effort to gain more information about other parties' inputs, intermediate results, or overall outputs by the transcripts of the protocol. An adversary can corrupt a party to receive all messages directed to it and control the messages to be sent out from it.

Herein, we denote the joint output of the ideal world adversary $\mathcal{S}$ and the outputs of the remaining parties in an ideal execution for computing the functionality $\mathcal{F}$ with inputs $\vec{x} = (x_1, x_2, \ldots, x_m)$ as $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\vec{x})$, the joint output of the real world adversary $\mathcal{A}$, and the outputs of the remaining parties in an execution of protocol Π with inputs $\vec{x} = (x_1, x_2, \ldots, x_m)$ as $\text{REAL}_{\Pi,\mathcal{A}}(\vec{x})$. Then, we say that protocol Π securely realizes functionality $\mathcal{F}$ if, for every real adversary $\mathcal{A}$ corrupting any parties and possibly the cloud servers, there exists an ideal world adversary $\mathcal{S}$ with black-box access to $\mathcal{A}$ such that, for all input vectors $\vec{x}$, $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi,\mathcal{A}}(\vec{x})$.

4. Our Result

We consider the secure outsourcing computation problem in the multiple users-two-cloud-servers scenario described as follows. There are $m + 2$ parties including m users and 2 noncolluding cloud servers: one is called the storing-cloud (SC), and the other is called the computing-cloud (CC). Each user P_i has a private input denoted as x_i and a pair of public-private keys (pk_i, sk_i) while sharing a private random r_i with SC that has a private number k_{sc}. CC has a private number k_{cc}. Users want to outsource the task of computing function $f(\cdot)$ on users' private inputs to the two cloud servers. They only provide the ciphertexts of the private data encrypted by a lattice-based encryption scheme under their different public keys and require the cloud servers to give the authorized users the result while keeping the security of the inputs. What is more, users wish that the cloud servers take charge of all of the computations related to the function $f(\cdot)$ and that there is noninteraction of users whatsoever so that the computation and communication complexities of each user are independent of the function to be computed. Herein, we deem that the two rounds of inevitable communications and a request from a user to the cloud servers for computing function $f(\cdot)$ are the three basic rounds of communication in this paper. Then, for each user, they expect that there are no other interactions at all between any user-to-user or user-to-server except the three basic rounds of communication. Furthermore, the computation complexity of each user depends on the encryption scheme it has used. The framework of our construction can be illustrated in Figure 1.

In this following section, we formally propose our solution denoted as protocol Π for convenience in detail and then analyze its security using the real-ideal paradigm in the semihonest model.

Without loss of generality, we represent the function $f(\cdot)$ to be computed by means of arithmetic circuit $\mathcal{C}_f$ consisting of any number of addition gates and l multiplication gates where each gate has two input wires and one output wire. Then, any functionality can be reduced to the two basic operations, addition and multiplication, over two inputs. Our construction can be summarized in Algorithm 2.

Protocol π: Two cloud servers-assisted secure outsourcing computation protocol

Setup.

For $i = 1, 2, \ldots, m$, sample a ring element vector $a_i \leftarrow R_q^n$, a ring element $s_i \leftarrow \chi$, and a ring element vector $x_i \leftarrow \chi^n$. Then, the private key of P_i is $sk_i := s_i$; the public key of P_i is $p_i := a_i s_i + 2x_i \in R_q^n$. And then, P_i shares a private random r_i with SC that has a private number k_{sc}. CC has a private number k_{cc}. As a preparation, user P_i firstly sends $r_i \cdot s_i$ to CC; CC computes $k_{cc} \cdot r_i \cdot s_i$ and then sends it back to SC. Then, SC can obtain $k_{cc} \cdot s_i$ by removing r_i.

Upload.

For $i = 1, 2, \ldots, m$, each user P_i encrypts its own private input x_i by the LE scheme. Firstly, P_i samples $e_i \leftarrow \chi^n$ and computes $c_0^i := \langle p_i, e_i \rangle + x_i \in R_q$ and $c_1^i := \langle a_i, e_i \rangle \in R_q$. Then, it outputs the ciphertext $c_i := \left(c_0^i, c_1^i\right) \in R_q^2$.

Outsourcing Computation.

After receiving all ciphertexts of the private inputs from users, SC stores all ciphertexts and executes a midtransformation to the outsourced data when computing some function $f(\cdot)$. After that, CC further transforms the midtransformed ciphertexts and then computes the function $f(\cdot)$ following the circuit $\mathscr{C}_f$ that consisted of addition gates and multiplication gates.

(1) Midtransforming.

Firstly, SC midtransforms the ciphertexts encrypted by users' own keys as $c_i \rightarrow c_i'$, where $c_i' = \left(c_0^{i\prime}, c_1^{i\prime}\right) = \left(k_{sc} \cdot c_0^i, k_{sc} \cdot \left(k_{cc} \cdot s_i\right) \cdot c_1^i\right)$, and sends c_i' to CC.

(2) Computing.

After receiving c_i', CC further transforms c_i' to $c_i'' = \left(k_{cc} \cdot k_{sc} \cdot c_0^i, k_{sc} \cdot \left(k_{cc} \cdot s_i\right) \cdot c_1^i\right)$. Denote $k = k_{sc} \cdot k_{cc}$; then, $c_i'' = \left(c_0^{i\prime\prime}, c_1^{i\prime\prime}\right) = \left(k \cdot c_0^i, k \cdot s_i \cdot c_1^i\right)$. CC then computes the ciphertext of the result by the transformed ciphertexts of users' private inputs.

Add. For each addition gate, $c_i'' \oplus c_j'' = \left(c_1^{i\prime\prime} - c_0^{i\prime\prime}\right) \oplus \left(c_1^{j\prime\prime} - c_0^{j\prime\prime}\right) = k \cdot \left(x_i + x_j\right)$.

Mul. For each multiplication gate, $c_i'' \otimes c_j'' = \left(c_1^{i\prime\prime} - c_0^{i\prime\prime}\right) \otimes \left(c_1^{j\prime\prime} - c_0^{j\prime\prime}\right) = k^2 \cdot \left(x_i \times x_j\right)$.

(3) Producing Custom-Made Result.

After computing gate by gate following the circuit $\mathscr{C}_f$, CC obtains the intermediate result encrypted by the private numbers of the two assisted cloud servers SC and CC; that is, $y' = k^{l+1} \cdot y = k_{sc}^{\ l+1} \cdot k_{cc}^{\ l+1} \cdot y$, where $y = f\left(x_1, x_2, \ldots, x_m\right)$ and l is the number of the multiplication gates of $\mathscr{C}_f$. To produce a custom-made result for each user, CC firstly sends y' to SC. SC removes k_{sc}^{l+1} and adds r_t to compute $y_t' = r_t \cdot k_{cc}^{l+1} \cdot y$ and then sends y_t' back to CC. CC finally removes k_{cc}^{l+1} to produce the custom-made ciphertext $y_t = r_t \cdot y$ and sends it to the authorized party P_t, $t \in \{1, 2, \ldots, m\}$.

Output

For each authorized party P_t, $t \in \{1, 2, \ldots, m\}$, it obtains the result y by removing r_t.

Algorithm 2

In setup, each user P_i invokes KeyGen(1^k) to compute its public-private keys (pk_i, sk_i). At the same time, each P_i selects a random r_i and sends it to SC via secure channels, while SC and CC, respectively, choose private numbers k_{sc} and k_{cc}. Assuming that all users' private data x_i, $i = 1, 2, \ldots, m$, are the real inputs of function $f(\cdot)$, then each P_i sends $r_i \cdot s_i$ to CC. CC further computes $k_{cc} \cdot r_i \cdot s_i$ and sends it to SC. After that, P_i submits the ciphertext c_i of the private input x_i encrypted by its own public key p_i to SC in the upload process.

In computation process, SC firstly midtransforms the outsourced data which are encrypted by different keys of users, and CC further transforms the midtransformed ciphertexts to the ones that are blinded by the same private numbers of the two servers so that CC can operate on the ciphertexts to compute $f(\cdot)$. Specifically, for addition/multiplication gate, CC can easily get the result by $(c_1^{i\prime\prime} - c_0^{i\prime\prime}) \oplus (c_1^{j\prime\prime} - c_0^{j\prime\prime}) = k \cdot (x_i + x_j)$ and $(c_1^{i\prime\prime} - c_0^{i\prime\prime}) \otimes (c_1^{j\prime\prime} - c_0^{j\prime\prime}) = k^2 \cdot (x_i \times x_j)$. Computing gate by gate following the circuit $\mathscr{C}_f$, CC can obtain the intermediate result $y' = k^{l+1} \cdot y = k_{sc}^{l+1} \cdot k_{cc}^{l+1} \cdot y$.

In order to guarantee that only the authorized user can get the final result, SC and CC cooperatively produce a custom-made result for each authorized user as follows. (Herein, we assume that P_t, $t \in \{1, 2, \ldots, m\}$, is authorized to get the result.) Firstly, CC sends y' to SC. SC removes k_{sc}^{l+1} and adds r_t to compute $y_t' = r_t \cdot k_{cc}^{l+1} \cdot y$ and then sends y_t' back to CC. CC finally removes k_{cc}^{l+1} to obtain the custom-made ciphertext $y_t = r_t \cdot y$ and sends it to P_t.

In the last process, the authorized user P_t obtains the result y by removing r_t from y_t.

5. Analysis

From the protocol described above, the correctness is obvious due to the homomorphic properties of the transformed ciphertexts. We will have a detailed discussion on its security. Note that, before the actual computations which are performed by SC and CC, there are setup and upload processes. We will individually illustrate their security at first. Afterwards, we will prove the security of the core of our protocol, that is, the outsourcing computation process, in the real-ideal framework. Finally, from the composition theorem [5], we can conclude that our protocol is secure.

Theorem 1. *Protocol* Π *is secure as long as the LE scheme is secure and SC and CC are noncolluding.*

Proof. Firstly, we look at the setup and upload processes individually.

In setup, each user, respectively, encrypts its private input by its own public key which is produced by invoking a semantically secure LE scheme. The security of this process is obvious. Afterwards, P_i sends $r_i \cdot s_i$ to CC and CC sends $k_{cc} \cdot r_i \cdot s_i$ to SC. Herein, P_i's private key s_i is protected by the blinding factors: r_i which is private to P_i and SC and k_{cc} which is private to CC. Therefore, the private keys of users will not be revealed in this process.

In upload, users outsource the encrypted data to SC. Since the LE scheme is semantically secure, given two ciphertexts $c_i(m_1)$, $c_i(m_2)$ of the two plaintexts m_1, m_2 uploaded by P_i, it is computationally infeasible for SC to distinguish the two ciphertexts. Hence, users can store their encrypted data in SC securely.

In outsourcing computation process, SC firstly midtransforms c_i to c_i' and sends c_i' to CC. It is obvious that it is secure since SC blinds the midtransformed ciphertext c_i' by the private number k_{sc}, which is secret to CC. As to the core of the computation process, we will discuss the security in the real-ideal framework. From the security definition, we say that protocol Π is secure if all adversarial behavior in the real world can be simulated in the ideal model where there exists an additional trusted party to perform all computations related to the function $f(\cdot)$ to be computed. We assume that there is a simulator $\mathcal{S}$ in the ideal world and then prove that it can simulate the semihonest adversary $\mathcal{A}$ that exists in the real execution. Since CC is able to independently complete addition and multiplication operations, we only need to prove that Add and Mul are secure against the semihonest adversary $\mathcal{A}$ corrupting CC. We prove this as follows.

Simulator $\mathcal{S}$. Run $\mathcal{A}$ on input$\{c_{\mathcal{S}}(m_1), c_{\mathcal{S}}(m_2)\}$.

Firstly, $\mathcal{S}$ computes

$$\begin{aligned} c_{\mathcal{S}}(m_1) &= \mathrm{Enc}(pk_{\mathcal{S}}, 1); \\ c_{\mathcal{S}}(m_2) &= \mathrm{Enc}(pk_{\mathcal{S}}, 1) \end{aligned} \tag{1}$$

and sends $c_{\mathcal{S}}(m_1)$, $c_{\mathcal{S}}(m_2)$ to $\mathcal{A}$.

Secondly, $\mathcal{A}$ sends two ciphertexts $c_{\mathcal{S}}(m_1^*)$, $c_{\mathcal{S}}(m_2^*)$ to $\mathcal{S}$. Then, $\mathcal{S}$ computes

$$\begin{aligned} c_{\mathcal{S}}(m_1^* + m_2^*) &= c_{\mathcal{S}}(m_1^*) \oplus c_{\mathcal{S}}(m_2^*); \\ c_{\mathcal{S}}(m_1^* \times m_2^*) &= c_{\mathcal{S}}(m_1^*) \otimes c_{\mathcal{S}}(m_2^*) \end{aligned} \tag{2}$$

and returns $c_{\mathcal{S}}(m_1^* + m_2^*)$, $c_{\mathcal{S}}(m_1^* \times m_2^*)$ to $\mathcal{A}$.

Finally, $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

Now, we can prove the security of Add and Mul algorithms by contradiction. Firstly, we assume that the view of the adversary $\mathcal{A}$ in the real world is distinguishable from the view simulated by the simulator $\mathcal{S}$. Then, we could find an algorithm to distinguish the ciphertexts encrypted by the LE encryption scheme, which is contrary to our assumption that the LE is semantically secure. Hence, the view of the adversary $\mathcal{A}$ in the real world is indistinguishable from the view simulated by the simulator $\mathcal{S}$. That is,

$$\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}}(c_{\mathcal{S}}(m_i)) \overset{c}{\approx} \mathrm{REAL}_{\Pi,\mathcal{A}}(c_{\mathcal{S}}(m_i)), \quad i = 1, 2. \tag{3}$$

Therefore, the two algorithms Add and Mul are secure. Furthermore, from the composition theorem [5], we can conclude that our protocol is secure as long as the LE scheme is secure and SC and CC are noncolluding in semihonest scenario. □

6. Conclusion

Only contributing the encrypted forms of their private inputs under their own public keys to gain the desired result of some function on the private inputs via powerful cloud with minimal computations and communications is the optimal method especially when users want to compute some complex function. In this paper, we introduce two noncolluding cloud servers to construct a secure outsourcing multiparty computation protocol on lattice-based encrypted cloud data under multiple keys in semihonest scenario. All computations related to the computing function are in the charge of the two cloud servers. Therefore, the computation complexity of each user only depends on the encryption scheme it has used. What is more, the communication complexity of each user is also independent of the function to be computed and there is no interaction of users whatsoever any more.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the NSFC (Grants nos. 61300181, 61272057, 61202434, 61170270, 61100203, and 61121061) and the Fundamental Research Funds for the Central Universities (Grants nos. 2012RC0612 and 2011YB01).

References

[1] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 160–164, Chicago, Ill, USA, 1982.

[2] Y. Lindell and B. Pinkas, "A proof of security of yao's protocol for two-party computation," *Journal of Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[3] O. S. Goldreich, S. Mical, and A. Wigderson, "How to play any mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing (STOC '87)*, pp. 218–229, New York, NY, USA, 1987.

[4] O. S. Goldreich, "Secure multiparty computation," Manuscript, Preliminary version, 1998.

[5] O. S. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*, Cambridge University press, 2004.

[6] M. M. Prabhakaran and A. Sahai, Eds., *Secure Multiparty Computation*, IOS Press, 2013.

[7] R. Fagin, M. Naor, and P. Winkler, "Comparing information without leaking it," *Communications of the ACM*, vol. 39, pp. 77–85, 1996.

[8] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols," in *Proceedings of the 20th annual ACM symposium on Theory of computing (STOC '88)*, pp. 11–19, 1988.

[9] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology—CRYPTO 2012*, vol. 7417 of *Lecture Notes in Computer Science*, pp. 643–662, Springer, 2012.

[10] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Advances in Cryptology—EUROCRYPT 2007*, vol. 4515 of *Lecture Notes in Computer Science*, pp. 52–78, Springer, 2007.

[11] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Advances in Cryptology—ASIACRYPT 2009*, vol. 5912 of *Lecture Notes in Computer Science*, pp. 250–267, Springer, 2009.

[12] M. Armbrust, A. Fox, R. Griffith et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[13] T. Velte, A. Velte, and R. Elsenpeter, *Cloud Computing, a Practical Approach*, McGraw-Hill, 2009.

[14] J. Loftus and N. P. Smart, "Secure outsourced computation," in *Progress in Cryptology—AFRICACRYPT 2011*, pp. 1–20, Springer, Berlin, Germany, 2011.

[15] M. J. Atallah and K. B. Frikken, "Securely outsourcing linear algebra computations," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communication Security (ASIACCS '10)*, pp. 48–59, April 2010.

[16] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-Party computation," *IACR Cryptology ePrint Archive*, vol. 2011, article 272, 2011.

[17] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IACR Cryptology ePrint Archive*, vol. 2013, article 13, 2013.

[18] B. Wang, M. Li, M. Chow et al., "Computing encrypted cloud data efficiently under multiple keys," in *Proceedings of the IEEE Conference on Communications and Network Security (CNS '13)*, pp. 504–513, 2013.

[19] M. Van Dijk and A. Juels, "On the impossibility of cryptography alone for privacy-preserving cloud computing," in *Proceedings of the 5th USENIX conference on Hot topics in security (HotSec '10)*, pp. 1–8, 2010.

[20] C. Gentry, *A fully homomorphic encryption scheme [Doctoral dissertation]*, Stanford University, 2009.

[21] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold fhe," in *Advances in Cryptology—EUROCRYPT 2012*, Lecture Notes in Computer Science, pp. 483–501, Springer, Berlin, Germany, 2012.

[22] S. Halevi, Y. Lindell, and B. Pinkas, "Secure computation on the web: computing without simultaneous interaction," in *Advances in Cryptology—CRYPTO 2011*, pp. 132–150, Springer, 2011.

[23] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS '11)*, pp. 97–106, 2011.

[24] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "Cloud-assisted multiparty computation from fully homomorphic encryption," *IACR Cryptology ePrint Archive*, vol. 2011, article 663, 2011.

[25] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Advances in Cryptology—CRYPTO 2011*, vol. 6841 of *Lecture Notes in Computer Science*, pp. 505–524, 2011.