Copyright ⓒ 2010 KSII

Semi-trusted Collaborative Framework for Multi-party Computation

Kok Seng Wong and Myung Ho Kim
School of Computing, Soongsil University
156-743 Sangdo-dong, Dongjak-Gu, Seoul Korea
[e-mail: {kswong, kmh}@ssu.ac.kr]
*Corresponding author: Kok Seng Wong

Received February 24, 2010; revised April1 10, 2010; accepted May 22, 2010; published June 30, 2010

Abstract

Data sharing is an essential process for collaborative works particularly in the banking, finance and healthcare industries. These industries require many collaborative works with their internal and external parties such as branches, clients, and service providers. When data are shared among collaborators, security and privacy concerns becoming crucial issues and cannot be avoided. Privacy is an important issue that is frequently discussed during the development of collaborative systems. It is closely related with the security issues because each of them can affect the other. The tradeoff between privacy and security is an interesting topic that we are going to address in this paper. In view of the practical problems in the existing approaches, we propose a collaborative framework which can be used to facilitate concurrent operations, single point failure problem, and overcome constraints for two-party computation. Two secure computation protocols will be discussed to demonstrate our collaborative framework.

Keywords: Data sharing, privacy preserving, secure computation protocol, collaborative computation

This work was supported by the Soongsil University Research Fund.

DOI: 10.3837/tiis.2010.06.013

1. Introduction

Collaborative computing is now the trend in most industries particularly in banking, finance, insurance, and healthcare. Most of the collaborative tasks are performed with internal or external parties. When two or more collaborators within a collaborative framework want to jointly perform a collaborative task, they need to share their private data with their counterparts. Privacy is frequently discussed during the development of collaborative systems under distributed environment. The balance between privacy protection and data sharing among collaborators is now becoming crucial. A secure system cannot guarantee that the privacy of the shared data is being preserved. However, a privacy preserved system normally can ensure that it has a series of secure mechanisms for privacy protection.

For distributed environments, much work has been focused on designing specific information sharing protocols [1]. However, the privacy of the shared data is becoming a challenging issue. In the data sharing operation, privacy is referred as the process to prevent data dissemination instead of the integration of privacy constraints [2].

There are two important considerations before any collaborative task can be performed. If privacy is not a major concern, each collaborator can send their private data to a Trusted Third Party (TTP). The TTP functions as the central repository or data warehouse to perform the collaborative tasks. This is an ideal approach to support most collaborative tasks if the data being shared is not sensitive information (e.g., sharing of project member's name, email address, contact numbers, and etc). If privacy is a concern, none of the collaborators should reveal their private data to any party including the TTP (e.g., sales analysis, product costs, stocks and etc).

It is clear that collaborative tasks are easy and straightforward if the privacy protection for those shared data is not taken into consideration. However, in most real life cases, the privacy concern cannot be ignored. The disclosure of private data could seriously and negatively impact the data owner. When datasets are distributed on different sites, there is a need to find a balance between security and privacy protection.

Database operations such as union or intersection computation, equijoin, and aggregation are important operations that can be used to support the secure data sharing process. For example, intersection computation is used to find the common value for different distributed datasets while revealing only the intersection [3]. However, the computation of these database operations have anonymity and security concerns [4]. Anonymity means the identity of the data owner should not be identified.

Let's consider a real life example where two companies A and B with shared customers would like to discover the buying behaviors of their customers. One of the interesting behaviors is to find the likelihood for a customer that buys product P_1 from A would also buy product P_2 from B. Due to the competition and business strategies, both companies do not agree to disclose their customer details to each other. In this situation, a secure and privacy protected framework is needed in order for A and B to mutually benefit.

Another real life example can be seen in health care related collaboration. Medical records for a single patient could be stored at different hospitals. Due to some local policies and privacy concerns, hospitals might refuse to share their patients' data with other parties. Without a secure and privacy protected framework, medical data sharing is difficult to implement.

In this paper, two secure computation protocols and a collaborative framework will be presented. We will particularly discuss secure summation protocol and set intersection in this paper. However, other sub-protocols such as scalar product computation and set union can utilize our collaborative framework.

In section 2, we discuss the background and related works in multi-party computation. We present our semi-trusted framework and secure computation protocols in sections 3 and 4. We give analysis on security, privacy and complexity of our protocols in section 5. Our conclusion is in section 6.

2. Background

When data is distributed in multiple locations, the access of private data by collaborators cannot be avoided. With privacy concerns, this data must be protected during the collaborative tasks. Whenever private data is being used, there will be a potential risk for privacy breaches. Several approaches and protocols have been proposed for data privacy and security protection.

2.1 Collaborative Works under Trusted Third Party

One of the direct implementations for multi-party computation is the usage of a trusted third party (TTP). Under this approach, all collaborators are connected to a central site. With the existence of a TTP, multi-party computation can be performed easily. Each collaborator reveals their private data to the TTP and allows the central site to compute the final result. The final result is then broadcast to all collaborators.

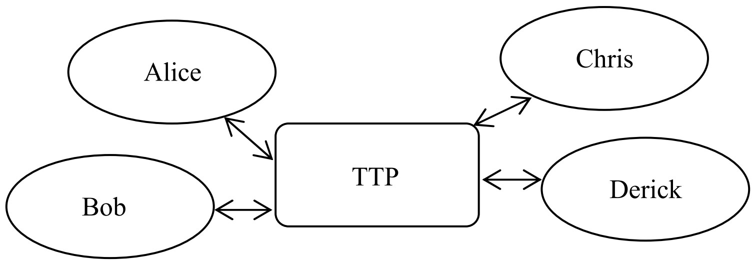

Fig.1. Trusted Third Party (TTP) Model.

As illustrated in **Fig. 1**, the TTP serves as central repository that will perform most of the operations. Construction of a central repository enables collaborators to share their data with other collaborators. However, this approach is viewed as insecure for collaborative works because the level of trust between collaborators and the trusted party is always unacceptable. The TTP model has been criticized to be too risky because it is always a single point target for malicious adversaries. It is an extremely valuable target for malicious parties because it holds most of the private data from the collaborators.

2.2 Secure Multi-party Computation

Secure Multi-party Computation (SMC) is an important method used to protect shared data. The concept of multi-party computation is to enable several parties jointly contribute their private data as the inputs for a specific computation function[1]. At the end of the computation, only the final result is revealed to all parties.

Secure two-party computation was first studied by Yao in [5]. In two-party computation, all computations were carried out by two parties without the involvement of others. Since then, many multi-party computation protocols have been proposed. As proved by Goldreich et al. in [6], there exists a secure solution for any functionality which can be represented as a combinatorial circuit. However, the circuit evaluation is somewhat inefficient for a large number of parties because the cost for large inputs can be very high.

The main objective of SMC is to perform secure computations in the absence of a TTP. However, without a TTP, current generic multi-party computations cannot support concurrent operations. Most of the approaches required collaborators to perform their operations in a sequential way. No parallel operations can be executed because each collaborator needs to wait for their direct neighbor to send them the data before they can perform their roles (e.g., data permutation, encryption, and etc).

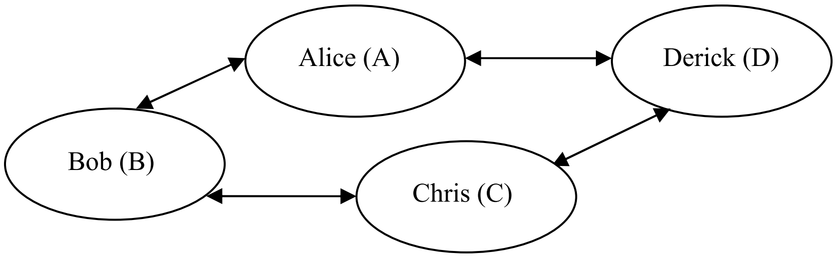

Fig. 2. Fully Secure Multi-party Computation Model.

In the absence of a TTP, a secure multi-party computation model is shown in **Fig. 2**. Assuming that there are 4 collaborators: Alice (A), Bob (B), Chris (C) and Derick (D), who would like to jointly compute for a specific function. A is the party who initiates the computation protocol. Here are some drawbacks in the general multi-party computation model:

- *It does not support concurrent operations.* No concurrent operations can be performed by all collaborators. For instance, B needs to wait for A (initiator) before it can start the computation. At the same time, D must wait for A and B before any operation can be performed.
- *Possible single point failure problem.* During the execution of the protocol, failure of any collaborator can cause the termination of the overall protocol. When collaborators are relying on each other, the computation can be terminated if one of them fails to perform their tasks. If the computation is aborted, the protocol needs to be restarted from the beginning.
- *Possible collusions among collaborators.* Since each collaborator has at least two direct connected neighbors, any two participants can collude together to find out the secret data of their common neighbor. For example, B and D can collude together to find the secret data of A or C.
- *Two-party computation problem.* If only two collaborators are involved in the computation, either party can derive the private data of their counterpart based on his own input and the final output.

[1] The computation functions can be a scalar product of two-party, set union of multi-party, and other set operations.

- *Chance to act maliciously*. In the existing design, all collaborators are connected in a ring. When the final output is broadcast to all collaborators, there exists a chance for any party to act maliciously by adding noise or alter the original result before sending it to the next party.
- *Unnecessary involvement*. In the case where the output is only required by a single party, some collaborators might still receive the output since they are connected in between the sender and the receiver. If any party ignored or failed to deliver the output, the computation protocol will fail.
- *Scalability and flexibility problem*. Once the protocol is started, disconnection of any participant will terminate the computation protocol. The arrangement of all participants is another issue that can limit the flexibility of the current SMC approach.
- *High execution costs*. When the size of the accumulated data is large, the communication and computation costs will increase.

2.3 Privacy Preserving Data Mining

Privacy Preserving Data Mining (PPDM) can be considered as an extension of the existing data mining process. Besides the knowledge extractions from a large volume of data, PPDM also ensures that all sensitive information is being protected. There are several approaches that have been proposed to solve the PPDM problems.

In 2000, Agrawal and Srikant proposed the first PPDM approach based on the data randomization and perturbation technique [7]. Since then, there have been many discussions and researches based on this technique [8][9]. As mentioned in [10], this technique is somewhat efficient, but there exists a tradeoff between data accuracy and privacy protection. When the data is strongly protected, the accuracy of the mining results cannot be accurate [11].

Cryptography based techniques, such as Secure Multi-party Computation (SMC) which was introduced in [12], have been widely studied in PPDM. Under this approach, a group of collaborators want to compute a function with their private inputs while keeping their inputs private. Only the final result to the query will be learned by the data miner, and no extra information should be revealed [6][13]. Lindell and Pinkas [14] examine the SMC problem based on cryptography techniques.

2.4 Summary

In the general design, participants for SMC or PPDM were connected to each other and formed a single continuous pathway. Each collaborator is connected directly with two or more neighbors. Outputs from both SMC and PPDM are correct and somewhat privacy protected. However, their design had some limitations which can cause significant problems in real world implementations. A typical SMC protocol required each party to “compute and pass” the accumulated result to the next party (as illustrated in **Fig. 2**). All parties have at least two direct communications with their direct neighbors.

When the size of the accumulated data becomes large, the communication and computation costs will increase. The computation cannot be performed simultaneously because each party needs to wait for their queue before they can start to compute and sends the data to the following party. If one of the parties fails to transmit the data, the overall computation will be stopped and it must be started from the beginning again.

2.5 Definitions and Notations

2.5.1 Definitions

Definition of Privacy: Under the collaborative framework, raw private data and all other permuted data should be protected. Raw data being shared from all collaborators should not be revealed to any party (internally or externally) before encryption. Only the data owner is able to access the raw data.

The final result needs to be protected such that no private data can be derived from it. In view of collaborator's privacy concerns, information such as identity, the number of collaborators, and where the sources come from should not be revealed. All collaborators perform their roles during the protocol execution without the knowledge of the existent of each other. The identity of the data miner (collaborator who initiates the operation) needs to be hidden from others.

Definition of Security: In terms of security, the protocol is secure if none of them are able to collude with each other. No direct communication is allow among collaborators. Both engines participate in the computation protocol without colluding with each other or with internal or external parties.

2.5.2 Notations

Table 1. Common notations used in the paper

Notation	Definition
DM	data miner
CE	computation engine
BE	broadcast engine
C_i	collaborator *i*
E_{pk}	encryption key from BE
D_{pk}	decryption key from BE
E_a	encryption key from party a
D_a	decryption key from party a
s_i	private input from collaborator *i*
$s_i^{'}$	first encryption by party *i*
$s_i^{''}$	second encryption by party *i*
$S^{''}$	aggregation of doubly-encrypted data
$S^{'}$	aggregation of encrypted data after first decryption by the broadcast engine

3. Semi-trusted Collaborative Framework

3.1 Framework Idea

The main idea behind our framework design is to provide a collaborative framework which can facilitate a secure data sharing mechanism and hence, to support multi-party computation. One of our main considerations is to allow collaborators to perform their roles independently and concurrently. Ideally, the best solution is by using a single trusted third party (TTP) as discussed in section 2.1. However, due to the privacy concerns, a single TTP is not suitable to be used in our design. One of the problems with a single TTP is that a completely reliable party is not always practical or available. We particularly consider the possibility of single TTP to collude with any collaborator to violate our privacy and security definitions in section 2.5. We utilized two semi-trusted engines in our framework, instead of a single TTP. Both semi-trusted engines participate in the protocol without learning any extra information or knowing the behavior of the collaborators. If both engines collude together, they should not learn anything. Our semi-trusted engines should follow the protocol and correctly execute the computation without colluding with each other or with any collaborator.

3.2 Framework Components and Design

In our proposed framework, we incorporate four components: a data miner (DM), broadcast engine (BE), computation engine (CE) and collaborators (C_i). Each component plays their role to ensure the computation protocol is executed in a secure environment and the privacy of shared data can be protected. The DM is one of the collaborators who initiates the protocol. The BE has two important roles during the protocol execution. First, it needs to validate all authorized collaborators and broadcasts the encryption key from the DM to all collaborators if the validation is successful.

Second, the BE is involved in the first decryption process. The CE is responsible in doubly-encrypted data computation by performing an additive operation in Protocol 1 or finding the set intersection in Protocol 2. Collaborators are parties who contribute their private data into the computation protocol. Any collaborator can play the role of the DM. We take the following assumptions for our framework design:

1. We assume that any collaborator who initiates the computation protocol will take the role as the DM. The encryption key from the DM is viewed as the broadcast key.
2. All collaborators need to be authenticated by the BE before they can participate in the framework. Only authorized collaborators will obtain the encryption key E_{pk} from the BE.
3. Both engines do not collude together to learn any private data or extra information during the execution of the protocol.
4. The protocol cannot be initiated by either the BE or the CE. Only authorized collaborators can initiate the protocol.
5. The BE must verify the DM as an authorized collaborator within the framework. If the verification process fails, the BE terminates the protocol immediately.
6. The DM needs to validate the broadcast key from the BE. If validation is fails, the DM replaces one of the encryption keys (E_{pk} or E_a) with another key.
7. The CE needs to terminate the protocol if the computation for doubly-encrypted data cannot be performed.
8. All decryption keys are private to their owner.

The design of our semi-trusted collaborative framework is shown in **Fig. 3**. We assume collaborator A initiates the protocol and takes the role as DM. In our framework, each collaborator has direct communication with the BE and the CE. The communications between collaborators and both engines are under a secure channel[2]. The details of the flow in our framework are as follows:

Step 1: The DM encrypts its encryption key (E_a) with E_{pk} and sends $E_{pk}(E_a)$ to the BE.

Step 2: The BE decrypts and broadcasts E_a to all collaborators (including the DM).

Step 3: Each collaborator performs double encryptions and sends the output to the CE.

Step 4: The CE performs secure computation and sends the output to the BE.

[2] A secure channel is a way of transferring data with resistant to interception and tampering.

Step 5: The BE decrypts and forwards the output to the DM for final decryption.

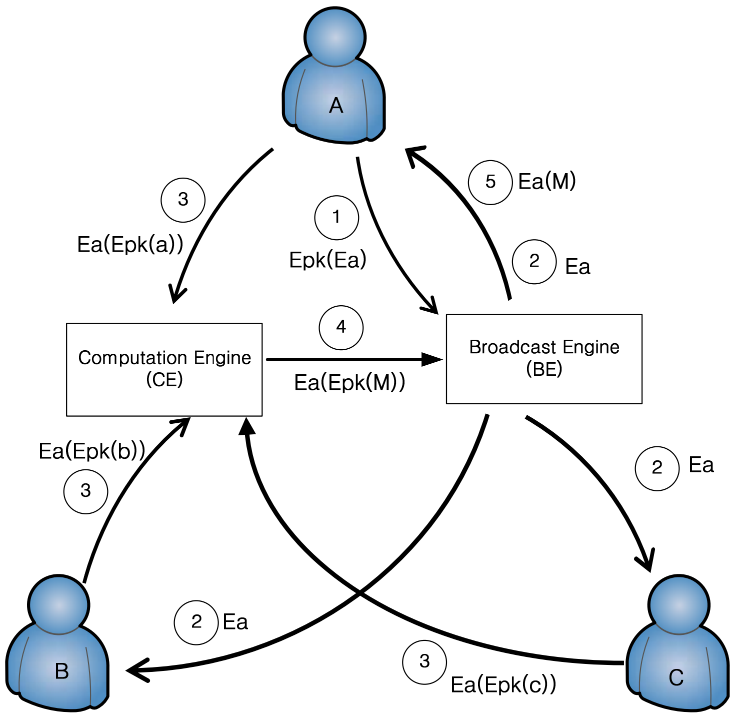

Fig. 3. Collaborative framework design. The numbers indicate the steps to perform the protocol.

In step 1, the DM initiates the computation protocol by sending $E_{pk}(E_a)$ to the BE. In step 2, the BE decrypts $E_{pk}(E_a)$ to obtain E_a. If the BE cannot decrypt $E_{pk}(E_a)$, it indicates that the DM is not an authorized collaborator and the protocol will be terminated. If the decryption operation is successful, the BE broadcasts E_a to all collaborators (including the DM). This step is important in order for the DM to verify that the BE is a valid engine used within the framework. If the broadcast key received is not E_a, the DM replaces one of the encryption keys (E_{pk} or E_a) used in the double encryption with another key.

Next, each collaborator performs double encryptions in step 3 with both encryption keys (E_{pk} and E_a) and sends the doubly-encrypted output to the CE. In step 4, the CE performs secure computation (i.e., secure summation and set intersection) and sends $E_aE_{pk}(M)$ to the BE. The protocol will terminate automatically if the CE fails to perform operations on the doubly-encrypted data. In step 5, the BE decrypts $E_aE_{pk}(M)$ and sends $E_a(M)$ to the DM. Finally, the DM decrypts $E_a(M)$ using D_a to obtain M.

3.3 Framework Communications

Communication among components is crucial to the security of our collaborative framework. We need to ensure that the computation protocol is secure while no extra information is revealed. In our design, the communication between the CE and the BE is restricted to one-way communication (CE → BE) as shown in **Fig. 3**.

When two or more collaborators are connected directly, there might be some chance for them to act maliciously. Communication between the DM and the BE is opened for two-way interactions. Both the DM and the BE need to have several interactions during the protocol execution including key exchanges, and broadcasting of encryption keys. The details will be discussed in a later section.

Table 2. Communication among components in the framework

Component	Communication	Direction
Data Miner (DM)	Two-way with BE	DM↔BE
Broadcast Engine (BE)	Two-way with DM One-way with C_i	BE↔DM BE→C_i
Collaborators (C_i)	One-way with CE	C_i→CE
Computation Engine (CE)	One-way with BE	CE→BE

The communication among components in our framework is shown in **Table 2**. The notation A↔B indicates two-way communication between A and B, while A→B indicates one-way communication from A to B. The communication flow of all components can be seen as DM↔BE→C_i→CE→BE. It is clear that all collaborators (including the data miner) will not have direct communication to all others.

4. Computation Protocol

4.1 Protocol Idea

The main idea behind our computation protocol is to allow collaborators to securely perform collaborative works and protect the shared data without revealing it to any party. Outputs during the protocol execution at each stage shall be protected and no extra information can be derived from the final result. The final result is accurate if each collaborator contributes their private input honestly. In most computation protocols, their focus is either on two-party or multi-party computation. The sub-protocols used for two-party or multi-party computation could be different or require substantial modifications on the existing protocol before it can be used to facilitate different environments. Our protocol shall be able to facilitate the computation for two-party or multi-party collaboration at the same time without changes in the framework design.

The design of computation protocol depends on the constraints and consensus among collaborators. All collaborators should be able to participate in the computation protocol. However, certain cases allow external parties to learn the result without contributing any input into the computation protocol. The final result might not be required by all contributors. Hence, it should not be broadcast to everyone. In this paper, we assume that the data miner is one of the collaborators who contributes their private data for the computation. The final result is only required by the data miner. We are going to use two computation protocols to demonstrate the implementation of our semi-trusted[3] collaborative framework. Secure summation can be used to demonstrate the steps involved in making a protocol secure [15]. We also show that other sub-protocols such as set intersection [16] can be adapted into our protocol easily.

4.2 Homomorphic Encryption Scheme

[3] Semi-trusted party only gathers information without modifying the behavior of the parties.

We will use the following additive homomorphism proposed by Paillier [17] in our secure computation: Let $E_a(m)$ denote the encryption of m with the encryption key, E_a. The scheme supports the following operations without the knowledge of the decryption key:

- Given two ciphertexts $E_a(m_1)$ and $E_a(m_2)$, there exists an efficient algorithm $+_h$ to compute

$$E_a(m_1) +_h E_a(m_2) = E_a(m_1 + m_2) \qquad (1)$$

- Given a constant c and a ciphertext $E_a(m_1)$, there exists an efficient algorithm $\cdot_h$ to compute

$$c \cdot_h E_a(m_1) = E_a(c \cdot m_1) \qquad (2)$$

- If m is encrypted with two different encryption keys, E_a and E_b

$$E_a(E_b(m)) = E_b(E_a(m)) \qquad (3)$$

4.3 Our Protocol

There are three phases in our computation protocol: preparation phase, encryption phase, and computation phase.

Preparation phase. In the preparation phase, the DM and the BE need to generate a cryptographic key independently. The encryption key from the BE is broadcast to all authorized collaborators before the protocol starts. The DM initiates the protocol by sending $E_{pk}(E_{ak})$ to the BE. Both parties exchange their encryption key while their individual decryption key is kept private and is not shared with others. Next, the BE broadcasts E_a to all other collaborators. At the end of this phase, each collaborator holds two encryption keys (E_a, E_{pk}) that will be used to encrypt their private data in the next phase.

```
Preparation Phase: Generation of cryptographic key pair by data miner
and broadcast engine.

for Broadcast Engine (BE)
    generates encryption key, $E_{pk}$
    broadcast $E_{pk}$ to all authorized collaborators
    broadcast $E_a$ if DM is an authorized collaborator
end for

for Data Miner (DM)
    generates encryption key, $E_a$
    sends $E_{pk}(E_a)$ to BE
end for
```

Encryption phase. Upon receiving the broadcast key from the BE, each collaborator performs doubly-encryption on their private data by using E_a and E_{pk}. Doubly-encrypted data will be sent to the CE.

Encryption Phase:

```
Each collaborator encrypts their private data with both encryption keys.
for each collaborator $C_i$ {concurrent operations} do
        for i=1 to k do
                $s_i^{'} = E_a(s_i)$ and $s_i^{''} = E_{pk}(s_i^{'})$
        end for
$C_i$ sends $s_i^{''}$ to Computation Engine (CE)
end for
```

Computation phase. In this phase, the CE receives *k* doubly-encrypted data from *k* collaborators. For a secure summation protocol, it performs additive operation to find the aggregation of all encrypted data without the decryption keys. The output will be forwarded to the BE. Next, the BE decrypts the received output and sends it to the data miner. The Data miner is able to find the final output because it holds the final decryption key.

```
Protocol 1: Secure Summation Protocol
Inputs: $s_1, s_2, s_3, ..., s_k \ (k \geq 2)$, where $s_i$ is the private input from each collaborator
Output: $S = \cup_{i=1}^{k} s_i$

Preparation Phase:
Generation of cryptographic key pair by data miner and broadcast engine.

Encryption Phase:
Each collaborator encrypts their private data with both encryption keys.

Computation Phase:
Computation engine aggregates all encrypted data.

for CE do
        for i=1 to k do
                $S^{''} = \cup_{i=1}^{k} s_i^{''}$
        end for
        Terminate the protocol if aggregation cannot be performed.
sends $S^{''}$ to BE
end for

for BE
        decrypts $S^{''}$ to obtain $S^{'} = \cup_{i=1}^{k} s_i^{'}$
Sends $S^{'}$ to DM
end for

for DM
        decrypts $S^{'}$ to obtains final output, $S = \cup_{i=1}^{k} s_i$
end for
```

```
for BE
        decrypts $S''$ to obtain $S' = \cup_{i=1}^{k} s_i'$
Sends $S'$ to DM
end for

for DM
        decrypts $S'$ to obtains final output, $S = \cup_{i=1}^{k} s_i$
end for
```

For the set intersection protocol, the CE finds the size of the intersection of all encrypted data. The CE does not require decryption keys to perform the computation because all data are doubly-encrypted with the same encryption keys. The final result can be sent directly to the DM or broadcast through the BE. If the final output is only required by the DM, we can remove step 4 in our framework design and replace it with a direct connection between the CE and the DM. The details of secure set intersection is shown in Protocol 2.

Protocol 2: Secure Set Intersection Protocol
Inputs: $s_1, s_2, s_3, ..., s_k (k \geq 2)$, where s_i is the private input from each collaborator
Output: $S = \left| \cap_{i=1}^{k} s_i \right|$

Preparation Phase:
Generation of cryptographic key pair by data miner and broadcast engine.

Encryption Phase:
Each collaborator encrypts their private data with both encryption keys.

Computation Phase:
Computation engine finds the intersection of all encrypted data.

```
for CE do
        for i=1 to k do
                $S = \left| \cap_{i=1}^{k} s_i'' \right| = \left| \cap_{i=1}^{k} s_i \right|$
        end for
sends S to BE
end for
```

4.5 Proof of Security and Correctness

The proof of security for our protocol depends on how much information has been revealed during the execution of the computation protocol. At the same time, our computation protocol should output a correct and accurate result.

Theorem 1: Additive operation performed by the CE in Protocol 1 is correct and precise without the need of decryption keys.

Proof: Based on Eq. 1, we can derive the additive operation for doubly-encrypted data as follow: Lets $x = E_b(m_1)$ and $y = E_b(m_2)$,

$$\begin{aligned} &E_a(E_b(m_1)) +_h E_a(E_b(m_2)) \\ &= E_a(x) +_h E_a(y) \\ &= E_a(x+y) \\ &= E_a(E_b(m_1+m_2)) \end{aligned} \tag{4}$$

As shown in Eq. 4, the CE does not required any encryption key in order to aggregate all encrypted data.

Theorem 2: Protocol 1 computes the summation of all private data (denoted as $S = \cup_{i=1}^{k} s_i$) without revealing extra information to any party.

Proof: Since each collaborator encrypts their private data with two encryption keys, no single party is able to decrypt the encrypted data. In our protocol, the CE aggregates all encrypted data without the knowledge of decryption keys. No party can see extra information during the execution of our protocol except the data miner at the end of the protocol. As for collaborators, they only participate in the encryption process without knowing other collaborators' identity. Both encryption keys must be used in this phase. Since collaborators do not have direct communication with each other, private data is secure and can only be viewed by its owner before the encryption operation. At the computation engine, all received data are in encrypted form (doubly-encrypted). The CE cannot see the private data because none of the decryption keys is known. Furthermore, the decryption process requires both decryption keys stored at the BE and the DM site. The BE will perform the first decryption process in the protocol. After the first decryption, the BE is not able to see the final result because the final decryption key is held by the DM.

Theorem 3: Assuming that all collaborators follow the protocol, the computation protocol correctly computes the summation of all private data, $S = \cup_{i=1}^{k} s_i$.

Proof: When each collaborator encrypts their private data with both encryption keys, each of them has $s_i^{''} = E_a(E_{pk}(s_i))$. The CE aggregates all doubly-encrypted data based on the Homomorphic Encryption Scheme in section 2.3.1. Hence, we have

$$\begin{aligned} S^{''} &= s_1^{''} + s_2^{''} + \ldots + s_{k-1}^{''} + s_k^{''} \\ &= E_a(E_{pk}(s_1)) +_h E_a(E_{pk}(s_2)) +_h \ldots +_h E_a(E_{pk}(s_{k-1})) +_h E_a(E_{pk}(s_k)) \\ &= E_a(E_{pk}(s_1 + s_2 + \ldots + s_{k-1} + s_k)) \end{aligned}$$

After the first decryption by the BE, we have

$$\begin{aligned} S^{'} &= D_{pk}(S^{''}) \\ &= E_a((s_1 + s_2 + \ldots + s_{k-1} + s_k)) . \end{aligned}$$

After the DM performs the last decryption, we will obtain the final result

$$S = D_a(S^{'})$$

$$= (s_1 + s_2 + \ldots + s_{k-1} + s_k)$$
$$= \cup_{i=1}^{k} s_i$$

The result from this protocol is correct.

Theorem 4: Assuming that all collaborators follow the protocol, the computation protocol correctly computes the set intersection of all private data, $S = \left|\cap_{i=1}^{k} s_i\right|$.

Proof: After each collaborator encrypts their private data with both encryption keys, the CE can determine the size of intersection for all received doubly-encrypted data. For any two doubly-encrytped data $s_i^{''}$ and $s_j^{''}$ (where i≠j), the CE can check the their intersection based on Eq. 3:

$$E_a(E_{pk}(s_i)) = E_a(E_{pk}(s_j)) \text{ if } s_i = s_j$$

Hence, the CE correctly computes the size of the intersection for all encrypted data.

5. Analysis and Discussions

In general, our protocol is able to support two-party or multi-party computation in a secure and privacy preserved environment. Since private data from each collaborator has been encrypted before they are used in the protocol, any party listening to the network traffic is not able to learn anything. All collaborators can perform concurrent operations during the execution of the protocol. In our design, each collaborator performs their operations independently without interference from others. They do not have to wait for others before they can perform their individual operation. Failure of any collaborator will not terminate the overall computation processes (single point failure problem). If any of the collaborators fails during the execution of our protocol, it will not stop others from continuing their operations.

5.1 Security Analysis

In our design, the BE can terminate the protocol if the DM is not an authorized collaborator. Once the verification process is successful, the BE will broadcast the encryption key from the DM to all collaborators. The DM is able to verify the BE based on the broadcast key it receives. The verification processes between the DM and the BE is essential in order to prevent malicious activities from parties who intend to interrupt the protocol. We considered two possible malicious attacks in our framework: (1) external attacks and (2) internal attacks.

External attacks involve external parties who are interested in learning some knowledge from the computation protocol. This knowledge may include the identity of collaborators, raw private data, intermediate results, and final output. Our collaborative framework is able to prevent external attacks such as a man-in-the-middle (MITM) attack[4]. In step 1, a malicious party, Eve may want to intercept E_a and replace it with her key E_a' . However, due to our framework design, it is impossible for Eve to verify herself as an authorized collaborator. The BE will terminate the protocol whenever the verification process fails. As such, Eve cannot continue the malicious attack in our framework.

[4] Man-in-the-middle attack (or known as bucket-brigade attack) is a form of eavesdropping where attacker impersonate to be the real component during the computation protocol.

Components within the framework may act maliciously or collude together for some malicious activities. This type of internal attack is more serious as compared to the external attacks because internal attackers have more knowledge about the private data, intermediate results, and computation outputs. It might be possible for the BE to act maliciously by substituting E_a with its key, E_a'. Then, the BE broadcasts $E_a^{'}$ to all collaborators. Collaborators will continue to participate in the protocol because they cannot determine whether the key received from the BE is generated by the DM. If the DM receives $E_a^{'}$ from the BE, it can conclude that the BE is performing a malicious attack. The DM cannot terminate the protocol at this point because other collaborators have already started to participate in the protocol. In order to prevent this attack, the DM replaces one of the encryption keys (E_{pk} or E_a) used in double encryptions with another key, E_b. In step 3, the CE receives doubly-encrypted data from all collaborators (including data from the DM). Since the DM encrypts his data with different sets of encryption keys, the CE will not be able to perform operations on the doubly-encrypted data. As shown in Theorem 1, additive operation for doubly-encrypted data is only possible if the same encryption keys were used in the encyption. If the CE cannot aggregate all doubly-encrypted data, it terminates the protocol immediately.

In our design, we do not allow direct communications among collaborators. This step is essential to prevent collaborators from colluding with each other. BE cannot collude with collaborators because of the one-way communication. Two engines being used in our framework to ensure that there is no single point of data compromise. It is not easy for a malicious adversary to control both engines at the same time.

5.2 Complexity Analysis

In terms of complexity, our protocol is determined by the communication and computation costs. The following contribute to the computation costs in our protocol: (1) Generation of cryptographic key pair by the data miner and broadcast engine. (2) The number of encryptions and decryptions in the computation and (3) Computation of additive homomorphic encryption by the computation engine.

5.3 Real World Implementation

Our collaborative framework is able to support collaborative works in a multi-party environment. Medical data sharing is one of the real world implementations which can utilize our collaborative framework design. The integration of information technology into the biomedical field has enabled practitioners and hospitals to collect, store, and analyze patient's medical records efficiently. Medical data sharing or information sharing has emerged as an important element in the health care industry to ensure the quality of health services. However, the level of correctness for shared medical data is important. By using our framework, hospitals (or other data owners) can act as collaborators while government agencies play the role of broadcast and computation engine.

6. Conclusions

In most of the secure computation protocols, collaborators need to participate in the computation. They are required to contribute local private data (either original or encrypted form) during the computation and take part in the computation. In some cases, it is not necessary for the data miner to reveal their private data when they are initializing the protocol. The final result is also not necessary to be broadcast to all participants at the end of the protocol. In this paper, we have focused on the collaborative framework for secure computation. Our collaborative framework can support two or more parties without revealing any extra information to any party. Compared with the existing approaches, our solution is more practical in terms of scalability and reliability in addition to security and privacy protection. At the end of the computation, only the data miner is able to view the final result. No extra information will be revealed. Other sub-protocols such as set union and scalar product can be easily implemented using our framework.

References

[1] R. Agrawal, A. Evfimievski, and R. Srikant, "Information Sharing across Private Databases," in *Proc. of 22nd ACM SIGMOD Int. Conference on Management of Data*, pp. 86-97, 2003.
[2] C. Clifton, M. Kantarcioglu, A. Doan, G. Schadow, J. Vaidya, A. K., Elmagarmid, and D. Suciu, "Privacy preserving data integration and sharing," in *Prof. of 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge*, pp. 19-26, 2004.
[3] M. Naor, and B. Pinkas, "Oblivious transfer and polynomial evaluation," in Proc. of *31st ACM Symposium on Theory of Computing*, pp. 245-254, 1999.
[4] B. Stefan, and O. Sebastian, "Secure set union and bag union computation for guaranteeing anonymity of distrustful participants," *Journal of Software*, vol. 3. no. 1, pp. 9-17, 2008.
[5] A. C. Yao, "Protocols for secure computations," in *Proc. of 23rd Symposium on Foundations of Computer Science*, pp. 160-164, 1982.
[6] O. Goldreich, S. Micali, and A. Wigderson, "How to Play Any Mental Game - A Completeness Theorem for Protocols with Honest Majority," in *Proc. of 19th ACM Conf. on Theory of computing*, pp. 218-229, 1987.
[7] R. Agrawal, and R. Srikant, "Privacy-preserving Data Mining," in *Prof. of 6th ACM SIGMOD Int. Conf. on Management of Data*, pp. 439-450, 2000.
[8] W. Du, Z. Zhan, "Using Randomized Response Techniques for Privacy-preserving Data Mining," in *Proc. of 9th ACM SIGKDD Conf. on Knowledge Discovery and Data mining*, pp. 505-510, 2003.
[9] A. Evfimievski, J. Gehrke, and R. Srikant, "Limiting Privacy Breaches in Privacy Preserving Data Mining," in *Prof. of 22nd ACM SIGMOD-SIGACTSIGART Symposium on Principles of Database Systems*, pp. 211-222, 2003.
[10] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar, "On the Privacy Preserving Properties of Random Data Perturbation Techniques," in *Proc. of 3rd ICDM IEEE Conf. on Data Mining*, pp.99-106, 2003.
[11] A. Evfimievski, R. Srikant, D. Agrawal, and J. Gehrke, "Privacy Preserving Mining of Association Rules," in *Proc. of 8th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pp. 217-228, 2002.
[12] A. C. Yao, "How to generate and exchange secrets," in *Proc. of 27th Annual Symposium on Foundations of computer science*, pp. 162-167, 1986.
[13] O. Goldreich, "The Foundations of Cryptography," *Cambridge University Press*, vo.2, 2004.
[14] Y. Lindell, and B. Pinkas, "Privacy Preserving Data Mining," *Journal of Cryptology*, vol. 15, no. 3, pp. 177-206, 2002.
[15] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, M. Y. Zhu, "Tools for Privacy Preserving Distributed Data Mining," *IEEE Trans. Knowledge Data Eng*ineering, vol. 16, no. 9, pp. 1026-1037, 2004.
[16] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Prof. of Eurocrypt 2004*, pp. 1-9, 2004.
[17] P. Paillier, "Public Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Proc. of Advances in Cryptography EUROCRYPT*, pp. 223-238, 1999.

Kok Seng Wong received the Bachelor of Computer Science in Software Engineering from University of Malaya, Malaysia in 2002 and received the Master of IT from Malaysia University of Science and Technology, Malaysia in 2004. Currently, he is working as lecturer in the School of Computing at Soongsil University. His research interests include data privacy and security, data sharing, privacy preserving data mining and knowledge management.

Myung Ho Kim received a Bachelor of Engineering in Computer Science from SoongSil University in 1989. He received a Master of Engineering and a Ph.D. in Computer Science from POSTECH, South Korea, in 1991 and 1995, respectively. He worked at ETRI, South Korea until Aug. 1995, and has been a Professor in the School of Computing at SoongSil University, Seoul, South Korea since Sep. 1995. He is a professor member of System Software Lab at Soongsil University. He specializes in Business Intelligence, Internet Security, and Distributed Computing.