IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 2, 2020, accepted June 13, 2020, date of publication June 18, 2020, date of current version June 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3003373

Privacy-Preserving Similarity Computation in Cloud-Based Mobile Social Networks

JUN ZHANG[1], SHIQING HU[1], AND ZOE LIN JIANG[2]
[1]Shenzhen University, Shenzhen 518060, China
[2]Harbin Institute of Technology, Shenzhen 518055, China

Corresponding author: Jun Zhang (jzhang3@cs.hku.hk)

This work was supported by the Research Start-up Fund of Shenzhen University.

ABSTRACT A growing number of mobile social network applications are taking advantage of cloud computing to store profiles of end users and run protocols which are compute-intensive. We focus on an application scenario of similarity computation between two users. To protect data security and privacy, mobile users encrypt their sensitive profiles before outsourcing to the cloud and different users choose different encryption keys. Three challenges need attention - how to compute on encrypted profiles under different keys, how to allow mobile users to stay offline during execution of the protocol, and how to select similarity metric. Existing schemes either rely on multi-key fully homomorphic encryption with one server or partially homomorphic encryption with two non-colluding servers. To balance computational complexity on one server and communication overhead between two servers, we put forward a privacy-preserving similarity computation protocol which supports both homomorphic additions and one homomorphic multiplication. We conduct security analysis and experimental evaluation of our scheme. The results show that our protocol is provably secure and runs reasonably fast, and thus can be applied in practice.

INDEX TERMS Privacy-preserving similarity computation, cloud computing, multiple keys, mobile social network.

I. INTRODUCTION

A social network provides an online platform for the individuals to build social relationships with other people who share similar interests, backgrounds or real-life connections (i.e., LinkedIn, Facebook). With the popularity of smart phones in recent years, an increasing number of mobile social network applications arise. *Similarity computation* is a required function in most social network applications. For instance, facebook users would like to know how similar they are to some potential new friends without disclosing their sensitive information.

A. BACKGROUND

Each mobile user in the social network is represented by a set of attributes. For example, user Alice has attributes {movie, song, piano, tennis} and user Bob has attributes {movie, tennis, basketball, dance}. To differentiate users' preferences, each attribute can be associated with a value to indicate how much they like this interest. A larger value means the user likes this interest more. To make it clear, Alice is denoted as {movie = 3, song = 4, piano = 3, tennis = 2}. Bob gets {movie = 5, tennis = 4, basketball = 5, dance = 2}. Through observing the profiles of Alice and Bob, we can know that they have two interests in common and Bob likes movie and tennis more than Alice. There exist different metrics to measure the similarity between mobile users such as Jaccard similarity, Dot product, Cosine similarity, Euclidean distance and so on. Jaccard similarity between sets A and B is defined as $J(A,B) = (|A \cap B|)/(|A \cup B|)$, which is the size of $A \cap B$ divided by the size of $A \cup B$. Dot product is the sum of the products of the corresponding entries of two vectors. Cosine similarity measures the cosine of the angle between two non-zero vectors, and is defined as $cos(\theta) = (\vec{a} \cdot \vec{b})/(||\vec{a}||||\vec{b}||)$ where $\vec{a}$ and $\vec{b}$ are vectors and θ is the angle between them. Euclidean distance between $\vec{a}$ and $\vec{b}$ is computed as $sqrt(\sum_{i=1}^{n}(\vec{a_i} - \vec{b_i})^2)$ where n is the dimension of vectors and *sqrt* means square root.

To protect user privacy, secure computation protocol must be carefully designed. Generally speaking, homomorphic encryption, secret sharing, oblivious transfer and garbled circuit are common privacy-preserving techniques.

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Aljawarneh.

Each technique has its own advantages and disadvantages [1]. For example, despite of low computational complexity, secret sharing incurs high communication overhead among the cloud servers. On the contrary, homomorphic encryption leads to high computational complexity but has low communication overhead. Different privacy-preserving techniques can also be combined together [2]. We focus on encryption-based schemes in this paper. The current privacy-preserving schemes for similarity computation [3], [4] have high computational complexity. Whereas, the computing capability of mobile phones is restricted due to the limitations of memory, battery and etc. As cloud computing consists of abundant and scalable resources, a promising trend is to make use of cloud computing to support mobile applications which involve extensive computation [5].

B. CHALLENGES

Three challenges exist if we want to incorporate cloud computing into mobile applications. First, a user loses control of his/her personal information once the sensitive data is outsourced to the cloud, which causes concern about data privacy and security. Second, mobile users should be allowed to stay offline when the cloud implements the protocol. Besides, the computational efficiency should be improved as much as possible to reduce latency. Third, it is hard to implement Jaccard or Cosine similarity on encrypted data, as they require division operation.

To cope with the first challenge, we choose *homomorphic encryption* to maintain data security. Homomorphic encryption allows computation on encrypted data and generates an encrypted output which is the same as being performed on decrypted data. Moreover, different users should encrypt their data with *different keys*. Otherwise, an individual's profile reveals to other users. To address the second issue, we should pay attention to *proxy re-encryption* which changes the encryption key of a ciphertext into a different one. To handle with the third problem, we choose Euclidean distance as the similarity metric in this paper. Euclidean distance is more accurate than dot product.

We propose a privacy-preserving similarity computation scheme for cloud-based mobile social networks with the existence of two non-colluding servers. Different users encrypt their profiles with different keys. Our contributions can be summarized as follows.

- Our scheme is fine-grained and we choose Euclidean distance as our similarity metric. We focus on partially homomorphic encryption (PHE) to improve efficiency. We present a method to make one additively homomorphic encryption support two keys. We also demonstrate how additively homomorphic encryption can be extended to compute one multiplication.
- Compared to existing PHE schemes, our scheme have competitive advantages. We abandon the ciphertext transformation step which changes all the encrypted profiles from different keys to one common key. Moreover, we remove interactions between the cloud servers when computing multiplication on additively homomorphic cipertexts.
- The mobile users need not to stay online after outsourcing their encrypted profiles to the cloud servers. With proxy re-encryption, the individual who launches a similarity computation request can stay offline until the cloud returns an encrypted result to him/her.

C. ORGANIZATION

The rest of this paper is organized as follows. Section II presents the literature review. Section III introduces our system model and threat model. Section IV provides some preliminary information. The details of our scheme are clarified in Section V. Then we perform security analysis in Section VI and experimental evaluation in Section VII. Finally, we conclude this paper in Section VIII.

II. LITERATURE REVIEW

A. SIMILARITY COMPUTATION SCHEMES

Existing privacy-preserving similarity computation schemes can be classified into two categories: *coarse-grained* and *fine-grained*. Coarse-grained approaches [3], [4] measure social similarity by counting the number of common attributes. To be precise, each user owns a set of social attributes. The *intersection* of two attribute sets indicates the social similarity between these two users. For example, user Alice has attributes {movie, song, piano, tennis} and user Bob has attributes {movie, tennis, basketball, dance}. The intersection between Alice and Bob is {movie, tennis} and the similarity can be denoted as 2, which is the size of the intersection. Supposed that we have another user Cindy who owns attributes {guitar, tennis, movie, hiking}, we can easily get the similarity between Alice and Cindy is 2 as well. However, *coarse-grained schemes cannot tell us whether Bob or Cindy is more similar to Alice*. Fine-grained similarity computation schemes [5]–[7] can solve this problem if each attribute is assigned a value to differentiate users' preferences. They use dot product between the profiles to measure the social similarity between two users.

We observe that coarse-grained schemes choose intersection of attribute sets as similarity metric while fine-grained schemes select dot product, both of which are based on homomorphic additions and multiplications. To compare dot product and Euclidean distance, we assume that user Alice has {movie $=$ 3, song $=$ 4, piano $=$ 3, tennis $=$ 2}, Bob has {movie $=$ 5, tennis $=$ 4, basketball $=$ 5, dance $=$ 2}, and user Cindy has attributes {guitar $=$ 4, movie $=$ 1, tennis $=$ 2, hiking $=$ 5}. The dot product between Alice ($< 3, 2 >$) and Bob ($< 5, 4 >$) is 21, and the dot product between Alice ($< 3, 2 >$) and Cindy ($< 1, 2 >$) is 7. Although the dot product between Alice and Bob is larger than Alice and Cindy, it cannot indicate Bob is more similar to Alice than Cindy. On the contrary, it is obvious that

Cindy is more similar to Alice than Bob, as they both have {tennis = 2}. The Euclidean distance between Alice and Cindy is smaller than the Euclidean distance between Alice and Bob. Therefore, we choose Euclidean distance as the similarity metric in this paper.

B. DIFFERENT ENCRYPTION KEYS

Different mobile users encrypt their profiles with different keys for the sake of security. In terms of the number of cloud servers, we differentiate the existing schemes that handle with different keys into two types. With one cloud server, multi-key fully homomorphic encryption algorithms [8], [9] are able to execute on encrypted data under different keys. However, the efficiency of these schemes is still an open question. Zhang *et al.* [10] use key-switching matrices to change different keys to a common key. Their scheme relies on a somewhat homomorphic encryption of which the efficiency is improved but is essentially single key. With two cloud servers, some solutions [11]–[13] are still single key in essence. They leverage a step called ciphertext transformation to convert the encrypted profiles from different keys to a common key. The interactions between the cloud servers are excessive [11], because every ciphertext is sent from the first server to the second server during ciphertext transformation. The second server then decrypts the ciphertext and re-encrypts it with a common key. There is a restriction between the two cloud servers that they should be non-colluding (i.e., no collusion between the two servers). The assumption of *two non-colluding servers* makes sense in the practical community [14]. Wang *et al.* [12], [13] make use of proxy re-encryption to transform ciphertexts on the first server without sending them to the second server. In this way, the interactions between the two servers are reduced. However, the underlying encryption of their scheme is only partially homomorphic. The communication overhead between the two servers is still substantial, as interactions are required to compute the operation that are not originally supported by the partially homomorphic encryption. The communication overhead is further reduced in [15] and the ciphertext transformation step is removed. They put forward a privacy-preserving regression protocol under different keys.

III. MODEL DESCRIPTION

A. SYSTEM MODEL

We present our system model in Figure 1. We list some mobile users named Alice, Bob, Cindy, Zoe, and etc. The cloud consists of two non-colluding servers, S1 and S2. Each mobile user has an individual profile and the mobile users' profiles are encrypted and outsourced to the cloud.

Supposed that Alice would like to know her similarity value with Bob, she will launch a similarity computation request. Then the cloud will run privacy-preserving similarity computation for Alice and Bob. We make an assumption that each mobile user in social network has a profile defined as a vector $< I_1, I_2, \ldots, I_i, \ldots, I_n >$ to indicate personal

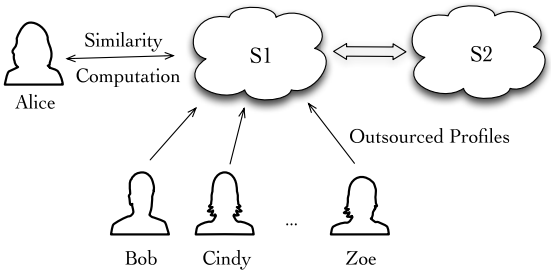

FIGURE 1. System model.

preferences, where n is the number of interests and I_i denotes a specific interest (i.e., music or reading). I_i takes value in the range of $[0, 5]$ for $i \in \{1, \ldots, n\}$. If $I_i = 0$, it means the user has no interest in the i_{th} hobby. On the other hand, $I_i = 5$ indicates the user extremely loves the i_{th} hobby. We use Euclidean distance between two vectors as the similarity metric.

B. THREAT MODEL

S1 and S2 are both honest-but-curious. They will follow the protocol step by step honestly, but they are curious about the contents of the encrypted profiles. S1 and S2 would never collude with each other. Mobile users are likely to collude with S1 instead of S2. There exist two possible attacks in our system. (i) A mobile user tries to know the content of encrypted profiles of other users. (ii) An attacker in the cloud (S1 or S2) tries to figure out the users' sensitive profiles by observing the input, intermediate results or final output.

IV. PRELIMINARIES

A. HOMOMORPHIC ENCRYPTION

Homomorphic encryption allows computation on encrypted data and generates an encrypted output which is the same as being performed on decrypted data. Homomorphic encryption can be divided into three categories: (i) Fully Homomorphic Encryption (FHE), (ii) Somewhat homomorphic encryption (SWHE), and (iii) Partially Homomorphic Encryption (PHE).

The first achievable FHE scheme was introduced by Gentry *et al.* [16], which allows any computable polynomial function to be executed on the encrypted data. Although it is a breakthrough, it is practically not a realistic scheme due to the heavy computation cost (bootstrapping part is especially costly). SWHE allows operations of addition or multiplication within a limited number of times (e.g., BGN [17]). PHE supports only one type of operation within an unlimited number of times (e.g., Paillier [18]). In other words, some PHE algorithms support only homomorphic additions while other PHE schemes support only homomorphic multiplications. PHE cannot support both homomorphic addition and homomorphic multiplication at the same time. Supposed that we are given two ciphertexts $E(m_1)$ and $E(m_2)$, we have $E(m_1 + m_2) = E(m_1)E(m_2)$ for additively homomorphic encryption (AHE). If we have another two ciphertexts $E'(m_1)$

Security Parameters: Given two safe primes p and q, we compute $N = pq$, $g = -a^{2N}$ $(a \in \mathbb{Z}^*_{N^2})$, the secret key $s \in [1, \frac{N^2}{2}]$, the public key $(N, g, h = g^s)$.

Encryption: To encrypt plaintext $m \in \mathbb{Z}_N$, we select a random $r \in [1, \frac{N}{4}]$ and generate the ciphertext $E(m) = (C_m^{(1)}, C_m^{(2)})$ as below:

$$C_m^{(1)} = g^r mod\ N^2 \quad and \quad C_m^{(2)} = h^r(1 + mN)\ mod\ N^2 \tag{1}$$

Decryption:

$$t = \frac{C_m^{(2)}}{(C_m^{(1)})^s} \qquad m = \frac{t - 1\ mod\ N^2}{N} \tag{2}$$

Single Key Homomorphism: Supposed that we have two plaintexts m_1, m_2 and their ciphertexts $E(m_1) = (C_{m_1}^{(1)}, C_{m_1}^{(2)})$ and $E(m_2) = (C_{m_2}^{(1)}, C_{m_2}^{(2)})$ under the same key s. The ciphertext $E(m_1 + m_2)$ can be computed as $E(m_1 + m_2) = (C_{m_1}^{(1)} C_{m_2}^{(1)}, C_{m_1}^{(2)} C_{m_2}^{(2)})$.

Proxy Re-encryption: If the secret key s is divided into two shares s_1, s_2 such that $s = s_1 + s_2$, then we can use s_1 to partially decrypt $E(m)$ to $E(m)' = (C_m^{(1)'}, C_m^{(2)'})$, which can be considered as a ciphertext under key s_2.

FIGURE 2. The BCP Cryptosystem.

and $E'(m_2)$, we get $E'(m_1 * m_2) = E'(m_1)E'(m_2)$ for multiplicatively homomorphic encryption (MHE). We use E and E' to emphasize they are different encryption algorithms, where E is additively homomorphic while E' is multiplicatively homomorphic.

In this paper, we focus on an additively homomorphic encryption method called BCP cryptosystem (also known as Modified Paillier Cryptosystem) [19]. We briefly review the BCP cryptosystem in Figure 2. Each ciphertext is made up of two parts and we have $E(m) = (C_m^{(1)}, C_m^{(2)})$, where the superscripts denote the first part and second part, respectively. The computation of $C_m^{(1)}$ and $C_m^{(2)}$ are shown in Equation (1). The decryption method is listed in Equation (2). To show the single key homomorphism of the BCP cryptosystem, assume that we have $E(m_1) = (g^{r_1}\ mod\ N^2, g^{sr_1}(1 + m_1 N)\ mod\ N^2)$ and $E(m_2) = (g^{r_2}\ mod\ N^2, g^{sr_2}(1 + m_2 N)\ mod\ N^2)$, then we get $E(m_1 + m_2) = E(m_1)E(m_2)$. The correctness of additive homomorphic property of BCP cryptosystem can be proved in Equation (3).

$$\begin{aligned} &E(m_1)E(m_2) \\ &= (g^{r_1+r_2}\ mod\ N^2, g^{s(r_1+r_2)}[1 + (m_1 + m_2)N]\ mod\ N^2) \\ &= E(m_1 + m_2) \end{aligned} \tag{3}$$

B. AHE SUPPORTS ONE MULTIPLICATION

Catalano and Fiore [21] showed a framework to enable existing additively homomorphic encryption (AHE) schemes to compute one multiplication. The idea is to transform a ciphertext $E(I_i)$ encrypted by an additively homomorphic encryption E into a "multiplication friendly" ciphertext $\mathcal{E}(I_i)$. To be specific, we have $\mathcal{E}(I_i) = (I_i - b_i, E(b_i))$ where b_i is a random number. Two non-colluding servers are used to store $\mathcal{E}(I_i)$ and b_i, respectively. For example, S1 stores $\mathcal{E}(I_i)$ and $S2$ stores b_i. Given two "multiplication friendly" ciphertexts $\mathcal{E}(I_1) = (I_1 - b_1, E(b_1))$ and $\mathcal{E}(I_2) = (I_2 - b_2, E(b_2))$, we compute multiplication $\mathcal{E}(I_1 I_2)$ as Equation 4.

$$\begin{aligned} \mathcal{E}(I_1 I_2) &= E[(I_1 - b_1)(I_2 - b_2)]E(b_1)^{I_2 - b_2} E(b_2)^{I_1 - b_1} \\ &= E(I_1 I_2 - b_1 b_2) \end{aligned} \tag{4}$$

To decrypt $\mathcal{E}(I_1 I_2)$, we will add $b_1 b_2$ to the decryption of $E(I_1 I_2 - b_1 b_2)$ where $b_1 b_2$ are retrieved from S2.

C. PROXY RE-ENCRYPTION

Proxy re-encryption [20] allows a third-party proxy to change the encryption key of a ciphertext to another key without decrypting. To understand how proxy re-encryption works, we introduce an application scenario of data-sharing. Provided that all the users encrypt their data and outsource the encrypted data to the cloud, if Alice wants to share data with Bob, a naive way is to download the encrypted data from the cloud first, decrypt the data and then send to Bob. Apparently, it is much better if cloud storage allows Alice to share data with Bob in a simpler way. Proxy re-encryption is able to remove redundant downloading and decrypting operations. Alice can generate a re-encryption key and send it to the cloud. Once the cloud obtains the re-encryption key, the cloud transforms Alice's encrypted data to what Bob can decrypt. Data security are guaranteed, as the cloud can only observe re-encryption key, Alice's encrypted data before and after transformation.

We also take the BCP cryptosystem as an example. According to the proxy re-encryption step in Figure 2, s_1 acts as a re-encryption key to achieve ciphertext transformation through partial decryption. We verify the correctness of proxy re-encryption in Equation (5). It is obvious that $E(m)'$ is a ciphertext encrypted by s_2.

$$\begin{aligned} E(m)' &= (C_m^{(1)}, \frac{C_m^{(2)}}{C_m^{(1)s_1}}) \\ &= (g^r\ mod\ N^2, \frac{g^{r(s_1+s_2)}(1 + mN)\ mod\ N^2}{g^{rs_1}\ mod\ N^2} \\ &= (g^r\ mod\ N^2, g^{rs_2}(1 + mN)\ mod\ N^2) \end{aligned} \tag{5}$$

V. OUR CONSTRUCTION

To handle with multiple keys, previous studies rely on ciphertext transformation which changes different keys to a common one. Ciphertext transformation causes high computational complexity, as it needs decryption and re-encryption. We achieve additive homomorphism under multiple keys by ciphertext extension. Compared with ciphertext transformation, ciphertext extension is simpler and runs more efficiently. Based on the framework described in Section IV-B, we are the first to construct a cryptosystem $\mathcal{E}_{BCP}$ to allow additively homomorphic encryption to support one multiplication under

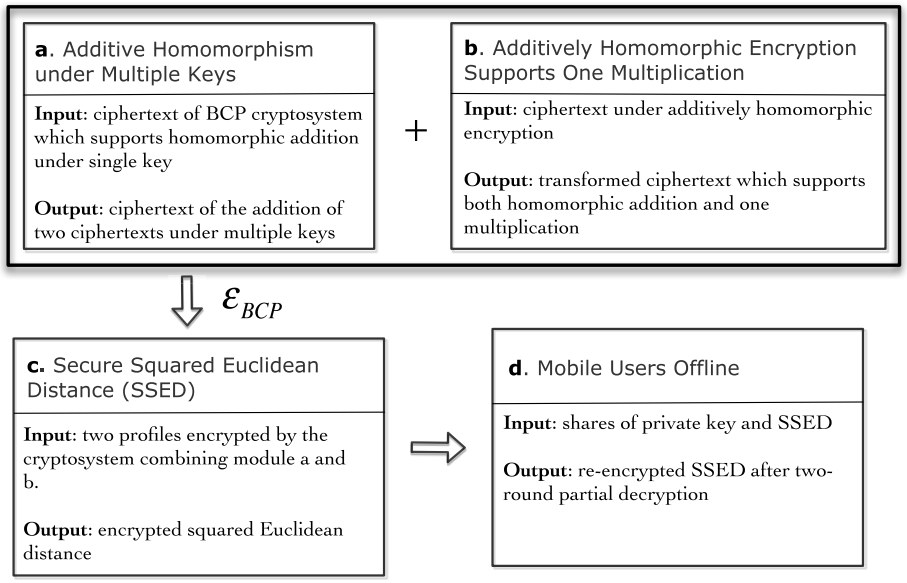

FIGURE 3. Building blocks.

multiple keys. Moreover, we choose Euclidean distance to measure the similarity between mobile users in a social network. As the square root function is not inherently supported by $\mathcal{E}_{BCP}$, we use squared Euclidean distance instead. Based on the $\mathcal{E}_{BCP}$ cryptosystem, we demonstrate how to compute secure squared Euclidean distance. Through sharing secret keys between two cloud servers, we successfully keep mobile users offline during the execution of our privacy-preserving protocol for similarity computation.

We show our building blocks in Figure 3. We need four modules denoted as **a**, **b**, **c**, **d**, respectively. Module **a** and **b** work together as our underlying cryptosystem which is additively homomorphic under multiple keys and supports one multiplication at the same time. Based on module **a** and **b**, we can compute secure squared Euclidean distance in module **c**. To keep mobile users offline, we require module **d** to run proxy re-encryption. In each module, we briefly highlight the input and output. For the details of our building blocks, please refer to Section V-A for additive homomorphism under multiple keys, Section IV-B for additively homomorphic encryption supports one multiplication under multiple keys, Section V-B for our $\mathcal{E}_{BCP}$ cryptosystem, Section V-C for secure squared Euclidean distance and Section V-D for mobile users offline.

A. ADDITIVE HOMOMORPHISM UNDER MULTIPLE KEYS

Recall that we introduce homomorphic encryption in Section IV-A and describe an additively homomorphic BCP cryptosystem in Figure 2. Given $E(m_1)$ and $E(m_2)$ under the same key s, $E(m_1 + m_2)$ can be computed as $E(m_1)E(m_2)$. We want to point out that the additive homomorphic property of BCP cryptosystem is based on single key, which means $E(m_1)$ and $E(m_2)$ are encrypted by the same key. Whereas, different mobile users encrypt their profiles with different keys to protect privacy in social network applications. We need an additive homomorphic encryption under multiple keys. To achieve this goal, we modify the ciphertext structure of BCP cryptosystem. A ciphertext of BCP crytosystem initially contains two parts, we have $E(m) = (C_m^{(1)}, C_m^{(2)})$. If we add another part to the original ciphertext, we can get the additive homomorphism under two keys. To make it clear, Alice has encrypted profile $E_a(I^a) = (C_{I^a}^{(1)}, C_{I^a}^{(2)})$ under key s_a and Bob has encrypted profile $E_b(I^b) = (C_{I^b}^{(1)}, C_{I^b}^{(2)})$ under key s_b. Then $E_{ab}(I^a + I^b)$ can be represented as $(C_{I^a}^{(1)}, C_{I^b}^{(1)}, C_{I^a}^{(2)} C_{I^b}^{(2)})$. As the ciphertext structure is altered, the decryption method should change accordingly (see Equation 6).

$$t = \frac{C_{I^a}^{(2)} C_{I^b}^{(2)}}{(C_{I^a}^{(1)})^{s_a} (C_{I^b}^{(1)})^{s_b}} \quad I^a + I^b = \frac{t - 1 \ mod \ N^2}{N} \tag{6}$$

If we want to obtain additive homomorphism under n keys, the ciphertext should be modified to contain $n + 1$ parts. One may argue that our ciphertext size increases linearly with the number of involved parties, which is unacceptable. We need to clarify that when Alice sends out a request of similarity computation to the cloud, we know that the cloud will calculate privacy-preserving Euclidean distance. Although different mobile users indeed encrypt personal profiles with different keys, we only need additive homomorphism under two keys. The reason is that when the cloud computes Euclidean distance between the encrypted profiles of each potential new friend and Alice, in fact, the cloud deals with two encryption keys each time. Therefore, the modified BCP ciphertext only contains three parts. The increase of the

size of ciphertext is negligible compared to the benefit that additive homomorphism brings under two keys.

B. AHE SUPPORTS ONE MULTIPLICATION UNDER MULTIPLE KEYS

We introduce a framework to enable existing additively homomorphic encryption schemes to compute one multiplication in Section IV-B. This framework has a nice property that it *inherits the multikey homomorphism* of the underlying additively homomorphic encryption.

In Section V-A, we describe how to modify the BCP cryptosystem to support additive homomorphism under two keys. In this section, we incorporate the additive homomorphism under multiple keys of the BCP cryptosystem into the framework proposed by [21] and obtain our final encryption scheme $\mathcal{E}_{BCP}$, where $\mathcal{E}$ denotes the framework and BCP denotes the underlying cryptosystem. To be specific, we assume that Alice has $\mathcal{E}_{BCP}(I_1) = (I_1 - b_1, E_a(b_1))$ and Bob has $\mathcal{E}_{BCP}(I_2) = (I_2 - b_2, E_b(b_2))$, where $E_a(b_1)$ and $E_b(b_2)$ are encrypted by s_a and s_b, respectively. We compute $\mathcal{E}_{BCP}(I_1 I_2)$ as Equation (7).

$$\begin{aligned}\mathcal{E}_{BCP}(I_1 I_2) &= E_a[(I_1 - b_1)(I_2 - b_2)]E_a(b_1)^{I_2 - b_2}E_b(b_2)^{I_1 - b_1} \\ &= E_{ab}(I_1 I_2 - b_1 b_2) \end{aligned} \tag{7}$$

C. SECURE SQUARED EUCLIDEAN DISTANCE (SSED)

Assume that Alice has encrypted profile $\mathcal{E}_{BCP}(I^a) = (I^a - b^a, E_a(b^a))$ under key s_a and Bob has encrypted profile $\mathcal{E}_{BCP}(I^b) = (I^b - b^b, E_b(b^b))$ under key s_b. The Euclidean Distance (ED) between I^a and I^b can be computed as follows.

$$ED(I^a, I^b) = \sqrt{\sum_{i=1}^{n}(I_i^a - I_i^b)^2} \tag{8}$$

The ciphertext of $(I_i^a - I_i^b)$ can be calculated as $\mathcal{E}_{BCP}(I_i^a - I_i^b) = [(I_i^a - I_i^b) - (b_i^a - b_i^b), E_{ab}(b_i^a - b_i^b)]$ where $E_{ab}(b_i^a - b_i^b)$ requires two-key homomorphism. According to Equation (7), we have

$$\mathcal{E}_{BCP}[(I_i^a - I_i^b)(I_i^a - I_i^b)] = E_{ab}[(I_i^a - I_i^b)^2 - (b_i^a - b_i^b)^2] \tag{9}$$

The secure squared Euclidean distance (SSED) is computed as Equation (10).

$$\begin{aligned} SSED(I^a, I^b) - E_{ab}[(b_i^a - b_i^b)^2] \\ = \sum_{i=1}^{n}\mathcal{E}_{BCP}[(I_i^a - I_i^b)(I_i^a - I_i^b)] \end{aligned} \tag{10}$$

D. MOBILE USERS OFFLINE

To make mobile users offline, we split each mobile user's secret key s into two shares. For example, Bob's secret key s_b is divided into two shares s_{b_1} and s_{b_2} and we have $s_b = s_{b_1} + s_{b_2}$. $S1$ holds s_{b_1} and $S2$ holds s_{b_2}. We rely on the proxy re-encryption property of $\mathcal{E}_{BCP}$, which inherits from the underlying BCP cryptosystem (see Figure 2). If we want to transform $\sum_{i=1}^{n} E_{ab}[(I_i^a - I_i^b)^2 - (b_i^a - b_i^b)^2]$ in Equation (10) to what Alice can decrypt, we should make S1 perform the first-round partial decryption with s_{b_1}, and then let S2 finish the second-round partial decryption with s_{b_2}. S2 sends $\sum_{i=1}^{n} E_a[(I_i^a - I_i^b)^2 - (b_i^a - b_i^b)^2]$ and $\sum_{i=1}^{n}(b_i^a - b_i^b)^2$ to Alice. Alice uses her own private key to decrypt the ciphertext and get her similarity value with Bob by computing $\sqrt{\sum_{i=1}^{n}(I_i^a - I_i^b)^2}$.

E. PRIVACY-PRESERVING SIMILARITY COMPUTATION

Based on the building blocks we introduced above, we put forward our privacy-preserving similarity computation protocol in Algorithm 1. The input is two encrypted profiles and the output is an encrypted similarity value. In step 1, Alice sends a request to the cloud to compute her similarity value with Bob. In step 2, cloud server S1 and S2 work together to compute the squared Euclidean distance securely (see Equation (10)). In step 3, S1 and S2 utilize proxy re-encryption to transform the encrypted Euclidean distance to what Alice can decrypt (see Section V-D). In step 4, S2 returns an encrypted result to Alice, which Alice will decrypt and obtain the final similarity value.

Algorithm 1 Protocol for Privacy-Preserving Similarity Computation

Input: Two encrypted profiles - $\mathcal{E}_{BCP}(I^a)$ and $\mathcal{E}_{BCP}(I^b)$.
Output: Encrypted similarity value.
1. Similarity Computation Request: Alice launches a request to compute her similarity value with Bob.
2. Secure Squared Euclidean Distance: The cloud server S1 computes secure squared Euclidean distance and get $SSED(I^a, I^b)$.
3. Proxy Re-encryption: S1 and S2 partially decrypt $SSED(I^a, I^b)$ to what Alice can decrypt with Bob's secret shares.
4. Similarity Computation Response: S2 sends $ED(b^a, b^b)$ and $E_a[SSED(I^a, I^b) - ED(b^a, b^b)]$ to Alice. Then Alice decrypts $E_a[SSED(I^a, I^b) - ED(b^a, b^b)]$ and calculates her similarity value $ED(I^a, I^b) = \sqrt{SSED(I^a, I^b)}$.

VI. SECURITY ANALYSIS

We consider the honest-but-curious model. All the mobile users, S1 or S2 follow our protocol step by step honestly, but they are curious to infer users' profiles by observing the inputs, outputs and intermediate results. There might exist collusion between a mobile user and S1. We perform security analysis of our scheme with the Real and Ideal paradigm and Composition Theorem [22]. We use a simulator in the ideal world to simulate the view of a semi-honest adversary in the real world. If the view in the real world is computationally indistinguishable from the view in the ideal world, the protocol is secure. According to the Composition Theorem, the entire scheme is secure if each step is proved to be secure.

Theorem 1: In Algorithm 1, it is computationally infeasible for S1 to distinguish the encrypted profiles of mobile users under different keys as long as $\mathcal{E}_{BCP}$ is semantically secure and the two servers are non-colluding.

Proof: We discuss the security of each step in Algorithm 1.

Step 1 (Similarity Computation Request): No sensitive profiles are involved in this step. It is straightforwardly secure.

Step 2 (Secure Squared Euclidean Distance): Given two encrypted profiles $\mathcal{E}_{BCP}(I^a) = (I^a - b^a, E_a(b^a))$ under key s_a and $\mathcal{E}_{BCP}(I^b) = (I^b - b^b, E_b(b^b))$ under key s_b, we have $\mathcal{E}_{BCP}(I_i^a - I_i^b) = [(I_i^a - I_i^b) - (b_i^a - b_i^b), E_{ab}(b_i^a - b_i^b)]$ and $\mathcal{E}_{BCP}(I_i^a - I_i^b)\mathcal{E}_{BCP}(I_i^a - I_i^b) = E_{ab}[(I_i^a - I_i^b)^2 - (b_i^a - b_i^b)^2]$. Computing $\mathcal{E}_{BCP}(I_i^a - I_i^b)$ requires additively homomorphic property. As a result, we should prove the security of addition over ciphertexts against a Semi-Honest adversary $\mathcal{A}_{S1}^{\mathbb{SH}}$ in the real world.

When calculating $\mathcal{E}_{BCP}(I_i^a - I_i^b)$, the view of $\mathcal{A}_{S1}^{\mathbb{SH}}$ in this step includes input $\mathcal{E}_{BCP}(I_i^a)$, $\mathcal{E}_{BCP}(I_i^b)$, and output $\mathcal{E}_{BCP}(I_i^a - I_i^b)$. We set up a simulator $\mathcal{F}^{\mathbb{SH}}$ in the ideal world to simulate the view of $\mathcal{A}_{S1}^{\mathbb{SH}}$. We assume that simulator $\mathcal{F}^{\mathbb{SH}}$ computes $\mathcal{E}_{BCP}(I_1)$ and $\mathcal{E}_{BCP}(I_2)$ where $I_1 = 1$ and $I_2 = 2$. Then the simulator computes $\mathcal{E}_{BCP}(I_1 + I_2)$ and returns $\{\mathcal{E}_{BCP}(I_1), \mathcal{E}_{BCP}(I_2), \mathcal{E}_{BCP}(I_1 + I_2)\}$ to $\mathcal{A}_{S1}^{\mathbb{SH}}$. Since the view of $\mathcal{A}_{S1}^{\mathbb{SH}}$ are ciphertexts encrypted by our $\mathcal{E}_{BCP}$ cryptosystem and $\mathcal{A}_{S1}^{\mathbb{SH}}$ does not know the corresponding encryption keys. If $\mathcal{A}_{S1}^{\mathbb{SH}}$ could distinguish the real world from the ideal world, then it indicates $\mathcal{A}_{S1}^{\mathbb{SH}}$ can distinguish ciphertexts generated by $\mathcal{E}_{BCP}$, which contradicts to the assumption that $\mathcal{E}_{BCP}$ is semantically secure. Therefore, $\mathcal{A}_{S1}^{\mathbb{SH}}$ is computationally infeasible to distinguish the real world from the ideal world, and addition over ciphertexts is secure.

For the case where a mobile user (i.e., Alice) colludes with S1, we use $\mathcal{A}_{(S1,Alice)}^{\mathbb{SH}}$ to denote the corresponding adversary. $\mathcal{A}_{(S1,Alice)}^{\mathbb{SH}}$ cannot learn anything beyond Alice's own profile and similarity with other mobile users.

Computing $\mathcal{E}_{BCP}(I_i^a - I_i^b)\mathcal{E}_{BCP}(I_i^a - I_i^b)$ requires multiplicatively homomorphic property. To prove the security of homomorphic multiplication, we observe that homomorphic multiplication is implemented by homomorphic additions as Equation (4) implies. Thus, as long as homomorphic addition is secure, we can conclude that homomorphic multiplication is also secure under the real and ideal paradigm. Furthermore, additive homomorphism under two keys is based on the ciphertexts of the fundamental BCP cryptosystem. As BCP cryptosystem is semantically secure, the homomorphic addition under two keys is secure as well.

Step 3 (Proxy Re-Encryption): To partially decrypt $SSED(I^a, I^b)$ with Bob's secret shares, S1 interacts with S2. Based on the hardness of computing discrete logarithm and the assumption that S1 and S2 does not collude, it is infeasible for S1 or S2 to recover Bob's secret key.

Step 4 (Similarity Computation Response): S2 sends $E_a[SSED(I^a, I^b) - ED(b^a, b^b)]$ and $ED(b^a, b^b)$ to Alice. As both b^a and b^b consist of random numbers, the attacker cannot infer anything from $ED(b^a, b^b)$. As $E_a[SSED(I^a, I^b) - ED(b^a, b^b)]$ is encrypted by Alice's key, Alice is the only person who can decrypt it and know the final similarity value. We make sure the individual who launches a request of similarity computation is the only authorized person that is allowed to obtain the matching result.

VII. EXPERIMENTAL EVALUATION

A. PERFORMANCE OF OUR SCHEME

The configuration of our PC is Windows 10 Enterprise 64-bit Operating System with Intel(R) Core(TM) i5-7500 CPU (4 cores), 3.41 GHz and 16 GB memory.

To provide platform independence, we use Java to implement our scheme together with open-source IDE. We use BigInteger class to process big numbers, which offers all the required basic operations. SecureRandom class is leveraged to produce a cryptographically strong random number. To generate safe prime numbers, we depend on the probablePrime method provided by BigInteger class. The probability that a BigInteger returned by this method is composite is below 2^{-100}. At the initialization period, public and private keys are generated. The time taken for generating a key pair varies with the bit-length of N. The performance of our scheme also relies heavily on the size of modulus N. We choose *three kinds of security parameters* - 1024-bit, 2048-bit, and 3072-bit. Obviously, we obtain higher security level with longer bit-length. However, it takes more time to encrypt users' profiles or compute multiplications on the ciphertexts when the bit-length increases. It is a balance between security and efficiency. We emphasize that 1024-bit and 2048-bit are popular choices and 3072-bit can be considered secure enough against various attacks nowadays. In our system model, we assume that each mobile user in social network has a profile defined as a vector $< I_1, I_2, \ldots, I_i, \ldots, I_n >$ to indicate personal preferences, where n is the number of interests and I_i denotes a specific interest. I_i takes value in the range of $[0, 5]$, which indicates how much an individual likes this interest.

To evaluate the performance of our scheme, we vary the dimension of a profile from 10 to 100. We show the encryption time in Table 1. When the bit-length is 1024, it takes 0.155 second to encrypt a profile of dimension 10 and 0.839 second to encrypt a profile of dimension 100. When the bit-length is 2048, it takes 0.632 second to encrypt a profile of dimension 10 and 6.223 seconds to encrypt a profile of dimension 100. When the bit-length is 3072, it takes 2.025 seconds to encrypt a profile of dimension 10 and 20.198 seconds to encrypt a profile of dimension 100. As indicated in Figure 4, the encryption time increases linearly with the dimension. Besides, the encryption time also increases if the bit-length increases.

To measure the performance of homomorphic addition under two keys, we run evaluations on three kinds of security parameters (1024-bit/2048-bit/3072-bit), and the results of running time for homomorphic addition are shown in Table 2.

TABLE 1. Encryption time under different parameters (second).

Dimension	1024-bit	2048-bit	3072-bit
10	0.155 s	0.632 s	2.025 s
20	0.186 s	1.248 s	4.051 s
30	0.257 s	1.915 s	6.053 s
40	0.336 s	2.550 s	8.079 s
50	0.422 s	3.127 s	10.190 s
60	0.502 s	3.750 s	12.107 s
70	0.585 s	4.394 s	14.159 s
80	0.669 s	4.986 s	16.330 s
90	0.753 s	5.676 s	18.219 s
100	0.839 s	6.223 s	20.198 s

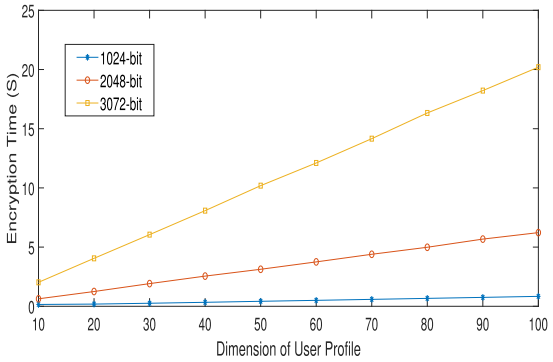

FIGURE 4. Encryption time.

TABLE 2. Running time for homomorphic addition (millisecond).

Dimension	1024-bit	2048-bit	3072-bit
10	0.263 ms	0.769 ms	0.872 ms
20	0.472 ms	1.049 ms	1.479 ms
30	0.497 ms	1.582 ms	2.173 ms
40	0.662 ms	2.091 ms	3.099 ms
50	0.816 ms	2.598 ms	3.302 ms
60	0.976 ms	3.189 ms	3.914 ms
70	1.125 ms	4.084 ms	4.845 ms
80	1.265 ms	4.423 ms	5.485 ms
90	1.426 ms	4.704 ms	5.856 ms
100	1.587 ms	5.18 ms	8.851 ms

When the bit-length is 1024, it takes 0.263 millisecond to encrypt a profile of dimension 10 and 1.587 milliseconds to encrypt a profile of dimension 100. When the bit-length is 2048, it takes 0.769 millisecond to encrypt a profile of dimension 10 and 5.18 milliseconds to encrypt a profile of dimension 100. When the bit-length is 3072, it takes 0.872 millisecond to encrypt a profile of dimension 10 and 8.851 milliseconds to encrypt a profile of dimension 100. We can observe that homomorphic addition runs very fast compared to encryption. Under different parameters, it all takes less tha 10 milliseconds to add up two profiles of dimension 100.

According to Equation (4) in Section IV-B, one homomorphic multiplication contains one encryption operation and two homomorphic addition operations. As the addition time only constitutes a small fraction, the homomorphic multiplication time depends largely on the encryption time. To improve the performance of homomorphic multiplications, we concentrate on accelerating the encryption procedure. As the acceleration of encryption helps speed up the computation of homomorphic multiplications, we observe the running time of each step in the encryption procedure (see Equation (1)). We find that computing g^r and h^r are time-consuming as they are exponential calculations. To make it clear, we take the 2048-bit encryption procedure as an example. The generation of random number r costs 557 ns (nanosecond). Whereas, it takes 54207 ns to compute $g^r \ mod \ N^2$ and 41757 ns to compute h^r. The calculation of $h^r(1 + mN) \ mod \ N^2$ consumes 191 ns. The public key of each mobile user is in the form of (N, g, h). We let the cloud server S1 generate random number r before receiving similarity computation requests, and thus the corresponding g^r and h^r can also be calculated in advance. In this way, we can greatly speed up the encryption process. To test the performance of multiplication on additively homomorphic encryption, we execute several instances. We present the results in Table 3. When the bit-length is 1024, it takes 20 milliseconds to run 100 homomorphic multiplications. When the bit-length is 2048, it takes 50 milliseconds to run 100 homomorphic multiplications. When the bit-length is 3072, it takes 90 milliseconds to run 100 homomorphic multiplications.

TABLE 3. Running time for homomorphic multiplication (millisecond).

Dimension	1024-bit	2048-bit	3072-bit
10	5 ms	10 ms	15 ms
20	6 ms	11 ms	20 ms
30	7 ms	19 ms	35 ms
40	10 ms	20 ms	45 ms
50	11 ms	30 ms	50 ms
60	13 ms	35 ms	55 ms
70	15 ms	40 ms	65 ms
80	16 ms	41 ms	75 ms
90	18 ms	43 ms	85 ms
100	20 ms	50 ms	90 ms

The computation of secure squared Euclidean distance (SSED) consists of homomorphic additions and multiplications. Given two profiles of dimension n, we need n homomorphic multiplications and $2n$ homomorphic additions to calculate the distance. Furthermore, homomorphic additions are executed on the ciphertexts of homomorphic multiplications. We show the running results of SSED calculation in Figure 5. When the bit-length is 1024, it takes 250 milliseconds (0.25 second) to compute SSED of two 100-dimension profiles. When the bit-length is 2048, it takes 895 milliseconds (0.895 second) to compute SSED of two 100-dimension profiles. When the bit-length is 3072, it takes 1709 milliseconds (1.709 seconds) to compute SSED of two 100-dimension profiles. On the other hand, we also record the running time of decrypting a secure squared Euclidean distance. To be precise, it takes 275 milliseconds and 561 milliseconds with and without proxy re-encryption to decrypt a SSED, respectively.

According to Section V-D, Bob's secret key s_b is divided into two shares s_{b_1} and s_{b_2} ($s_b = s_{b_1} + s_{b_2}$). $S1$ holds s_{b_1}

TABLE 4. Comparison between existing schemes.

	One Server	Two Servers	Coarse-grained	Fine-grained
[8]–[10]	✔	✘	**N/A**	**N/A**
[11]–[15]	✘	✔	**N/A**	**N/A**
[3], [4]	**N/A**	**N/A**	✔	✘
[5]	✘	✔	✘	✔
Our Scheme	✘	✔	✘	✔

N/A: Not Applicable

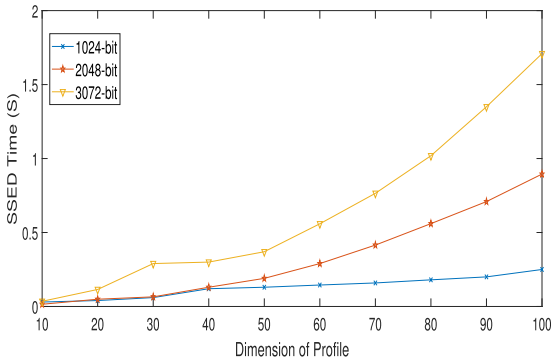

FIGURE 5. SSED time.

and $S2$ holds s_{b_2}. To transform a secure squared Euclidean distance to what Alice can decrypt, we should make S1 perform the first-round partial decryption with s_{b_1}, and then let S2 finish the second-round partial decryption with s_{b_2}. To measure the communication overhead between the two non-colluding servers, we also run experiments and record the size of interactions. When the bit-length is 1024, the communication overhead is 0.74 KB. When the bit-length is 2048, the communication overhead is 1.49 KB. When the bit-length is 3072, the communication overhead is 2.25 KB. As a result, our scheme is really competitive in terms of communication overhead, compared to those secret-sharing based or partially homomorphic encryption based schemes.

B. COMPARISON WITH EXISTING SCHEMES

We compare our scheme with existing protocols that are related to privacy-preserving similarity computation in Table 4. Protocols in [8]–[10] focus on dot product computation and depend on one cloud server, which are related to the SSED computation in this paper. Reference [8], [9] are fully homomorphic and [10] are somewhat homomorphic. They cannot be classified by coarse-grained or fine-grained privacy-preserving similarity computation, as they have different application scenarios. Reference [11]–[15] relies on two non-colluding servers. They all use partial homomorphic encryption. The main difference between these schemes is the communication overhead between two cloud servers. Reference [3], [4] are coarse-grained distributed schemes (i.e., no cloud server involved). Reference [5] takes advantage of cloud computing and requires two non-colluding servers, which is the most relevant to our scheme. Gao *et al.* use dot product as the similarity metric. We choose Euclidean distance and get more accurate result. Moreover, the solution in [5] has to run ciphertext transformation to make sure that encrypted profiles are under a common key. Whereas, this *ciphertext transformation step causes redundant computational overhead*. Besides, their scheme is only additively homomorphic and each multiplication incurs interactions between the cloud servers. Our scheme abandons the ciphertext transformation step and removes interactions between the cloud servers when computing multiplication on additively homomorphic cipertexts.

VIII. CONCLUSION

In this paper, we focus on mobile social network applications supported by cloud computing. Particularly, we are interested to answer questions from mobile users that how similar they are to some potential new friends. Under the assumption that there exist two non-colluding servers, we proposed a privacy-preserving similarity computation protocol. We use Euclidean distance to measure the similarity. As computing privacy-preserving Euclidean distance requires homomorphic additions and multiplications, we construct an encryption method that supports additive homomorphism and one multiplicatively homomorphic operation under multiple keys. Based on proxy re-encryption, we successfully keeps the mobile users offline during the similarity computation process. Our scheme is proven to be secure and the experimental results highlight the practicability of our scheme. Our building blocks such as additive homomorphism under multiple keys are not restricted to the similarity computation application in this paper. They can be extended to other application scenarios that require multi-key homomorphic additions and one homomorphic multiplication. Our future work is to construct a privacy-preserving kNN algorithm that allows a mobile user pick out k potential new friends at the same time. Moreover, we aim to enhance our security model and design protocols that are secure against malicious attackers in the cloud.

REFERENCES

[1] T. B. Pedersen, Y. Saygin, and E. Savaş, "Secret sharing vs. encryption-based techniques for privacy preserving data mining," in *Proc. Eurostat*, vol. 176, 2007, pp. 45–135.

[2] D. Demmler, T. Schneider, and M. Zohner, "ABY—A framework for efficient mixed-protocol secure two-party computation," in *Proc. NDSS*, 2015, pp. 1–15.

[3] M. Li, S. Yu, N. Cao, and W. Lou, "Privacy-preserving distributed profile matching in proximity-based mobile social networks," *IEEE Trans. Wireless Commun.*, vol. 12, no. 5, pp. 2024–2033, May 2013.

[4] L. Zhang, X.-Y. Li, K. Liu, T. Jung, and Y. Liu, "Message in a sealed bottle: Privacy preserving friending in mobile social networks," *IEEE Trans. Mobile Comput.*, vol. 14, no. 9, pp. 1888–1902, Sep. 2015.
[5] C.-Z. Gao, Q. Cheng, X. Li, and S.-B. Xia, "Cloud-assisted privacy-preserving profile-matching scheme under multiple keys in mobile social network," *Cluster Comput.*, vol. 22, no. S1, pp. 1655–1663, Jan. 2019.
[6] L. Zhang, X.-Y. Li, Y. Liu, and T. Jung, "Verifiable private multi-party computation: Ranging and ranking," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 605–609.
[7] R. Zhang, J. Zhang, Y. Zhang, J. Sun, and G. Yan, "Privacy-preserving profile matching for proximity-based mobile social networking," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 9, pp. 656–668, Sep. 2013.
[8] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proc. 44th Symp. Theory Comput. (STOC)*, 2012, pp. 1219–1234.
[9] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 395–412.
[10] J. Zhang, X. Wang, S.-M. Yiu, Z. L. Jiang, and J. Li, "Secure dot product of outsourced encrypted vectors and its application to SVM," in *Proc. 5th ACM Int. Workshop Secur. Cloud Comput. (SCC)*, 2017, pp. 75–82.
[11] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 12, pp. 2046–2058, Dec. 2013.
[12] B. Wang, M. Li, S. S. M. Chow, and H. Li, "Computing encrypted cloud data efficiently under multiple keys," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2013, pp. 504–513.
[13] B. Wang, M. Li, S. S. M. Chow, and H. Li, "A tale of two clouds: Computing on data encrypted under multiple keys," in *Proc. IEEE Conf. Commun. Netw. Secur.*, Oct. 2014, pp. 337–345.
[14] J. Zhang, M. He, and S.-M. Yiu, "Privacy-preserving elastic net for data encrypted by different keys—With an application on biomarker discovery," in *Proc. IFIP Annu. Conf. Data Appl. Secur. Privacy*. Cham, Switzerland: Springer, 2017, pp. 185–204.
[15] J. Zhang, M. He, G. Zeng, and S.-M. Yiu, "Privacy-preserving verifiable elastic net among multiple institutions in the cloud," *J. Comput. Secur.*, vol. 26, no. 6, pp. 791–815, Oct. 2018.
[16] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput. (STOC)*, 2009, pp. 169–178.
[17] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. Theory Cryptogr. Conf.* Berlin, Germany: Springer, 2005, pp. 325–341.
[18] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1999, pp. 223–238.
[19] E. Bresson, D. Catalano, and D. Pointcheval, "A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications," in *Advances in Cryptology*. Berlin, Germany: Springer, 2003, pp. 37–54.
[20] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. EUROCRYPT*. Berlin, Germany: Springer, 1998, pp. 127–144.
[21] D. Catalano and D. Fiore, "Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 1518–1529.
[22] O. Goldreich, *Foundations of Cryptography: Basic Applications*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2004.

JUN ZHANG received the Ph.D. degree in computer science from The University of Hong Kong, in 2018. She is currently an Assistant Professor with the Department of Educational Technology, Shenzhen University. Her research interests include cloud security, privacy-preserving data mining, and genomic privacy.

SHIQING HU received the master's degree in computer science from Northwestern Polytechnical University, in 1989. He is currently a Professor with the Department of Educational Technology, Shenzhen University. His research interests include computer network engineering and application of new technology in education.

ZOE LIN JIANG received the Ph.D. degree from The University of Hong Kong, Hong Kong, in 2010. She is currently an Associate Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. Her research interests include cryptography and artificial intelligence.