Hindawi
Security and Communication Networks
Volume 2018, Article ID 1265352, 11 pages
https://doi.org/10.1155/2018/1265352

Research Article

Privacy Preservation for Friend-Recommendation Applications

Weicheng Wang,[1] **Shengling Wang**,[1] **and Jianhui Huang**[2]

[1] *College of Information Technology and Science, Beijing Normal University, Beijing, China*
[2] *Institute of Computing Technology, The Chinese Academy of Sciences, Beijing, China*

Correspondence should be addressed to Shengling Wang; wangshengling@bnu.edu.cn

Received 1 August 2018; Accepted 9 October 2018; Published 21 October 2018

Guest Editor: Yan Huo

Friend-recommendation applications as one kind of typical social applications can satisfy the social contact needs of different users and become tools for developing a social relationship. However, the privacy leakage has turned into an insurmountable obstacle to the market success of such applications. Existing privacy protection approaches for social applications either introduce untrusted third parties or sacrifice information accuracy. As for friend-recommendation applications particularly, the multihop trust chain and anonymous message methods still have a defect that the hacker can act as a user to acquire information. In this paper, we put forward the privacy protection mechanism based on zero knowledge without any privacy leakage to the application server. In detail, the server knows nothing about the user's information, but can still provide users with accurate information on friend recommendation. We also analyze the potential attack methods and propose the corresponding solution. Our simulation results verify the effectivity and efficiency of our scheme.

1. Introduction

With the growing popularity of smart-devices, such as mobile phones, laptops, and tablets, the social applications gradually play an essential role in people's daily life [1, 2]. Facebook has 1.65 billion users with 1 billion active users simultaneously, and Twitter has 600 million users. In China, Ten-cent QQ has 829 million active users, WeChat has over 700 million active users, etc. [3]. The social applications mostly satisfy the social contact needs of different users and become tools for developing a social relationship. They can provide services for people without the limitation of distance. People can find friends who have the same interests and hobbies in the world and communicate whenever and wherever they want.

As one kind of typical social applications, friend-recommendation application has a primary purpose that it helps users to find friends based on the information provided by the users so that the recommended friends can satisfy the requirements of the users. According to the user's interests and location, the application can recommend suitable people to the users. However, once a user sends his query to the server, it is possible that this information is sold to the third party. Furthermore, even if the encryption algorithm is perfect, the malicious people can still act as a user to acquire information from the server. Once a user finds its privacy is leaked by the social applications, he will stop using such applications. When a large number of users stop using them, it is easy to cause the collective panic and the trust crisis.

Existing privacy protection schemes for social applications either introduce untrusted third parties [4–6] or sacrifice information accuracy [7–11]. In the schemes of introducing a third party, the third party acts as the agent of the users to submit the query requests to the server and returns the query results from the server to the users. In this case, the privacy information scattered in various service providers is concentrated in a few third parties. Hence, the risk and scale of privacy leakage are increased sharply. Once the third party is hacked, the crisis of online incredibility will endanger the whole social network heavily. In addition, considering the schemes of sacrificing information accuracy to realize the privacy protection, the server cannot provide perfect services based on the user's blurred private information. Besides, with the help of big data analysis, it is possible that the obscured information can be inferred or restored. Some of the methods used for friend-recommendation applications particularly propose the multihop trust chain [12] or utilize anonymous message [13, 14]. However, the malicious hackers can still act as a user to acquire information.

Motivated by these, our paper aims to realize zero-knowledge privacy protection. Here, zero knowledge implies that the server knows nothing. Therefore, our aim is challenging, because on the one hand the server needs information from the users to provide services, but on the other hand such information needs to be kept secret from the server.

To realize the seemly conflicted aims, a privacy preservation framework is proposed for friend-recommendation applications, including the communication processes between several entities and data structures. Furthermore, we propose an encryption algorithm based on homomorphic encryption and grouping attributes, so that the users' information can be protected well. We propose a method of district limitation for matching operation which can decrease the possibility of information leakage. We give two possible decryption situations and illustrate the different operations in the two cases. Finally, the potential attack methods are analyzed and the corresponding solutions are proposed. Our simulation results verify the effectivity and efficiency of the proposed scheme.

The remainder of the paper is organized as follows. Section 2 is for the related work. A summary of the framework is presented in Section 3. Section 4 introduces the detailed processes of the algorithms including encryption, decryption, and matching. Some specific defense designs are introduced in Section 5. Section 6 reports the results of our numerical simulations. Our conclusions are summarized in Section 7, followed by the acknowledgment.

2. Related Work

In the social network system, privacy is a rather important issue [15–20]. There are several methods achieving privacy preservation aim, which are based on k-anonymity model. The basic idea of k-anonymity is to remove some features so that each item is not distinguishable among other k items. For data protection, it protects data at the cost of the original data quality [9]. For location-based services (typical social applications), it realizes the privacy protection through blurring user's locations [7, 8, 21].

Based on the k-anonymity, a clustering perturbation algorithm for privacy protection in social networks was proposed [9]. It considers preserving privacy of vertex properties and community structures simultaneously. The algorithm introduces a strategy of exchanging attributes between vertices with the same degree randomly to induce attackers to search for false targets and maintain the whole structure of the network. In 2014, Rongxing Lu et al. [22] proposed a privacy-preserving framework for the local-area mobile social application (PLAM). It employs a privacy-preserving request aggregation protocol with k-anonymity and l-diversity properties, without involving a trusted anonymizer server to keep user's preference private, and integrates unlinkable pseudo-ID technique to achieve user's identity privacy and location privacy.

However, k-anonymity model presents a problem of accuracy loss and attackers can still perform trajectory attacks. It is not suitable in situations where the requirement of information accuracy is extremely high. In addition, k-anonymity does not protect users privacy when they are in a densely populated area.

Besides, a trust-based friend-recommendation scheme used for privacy-preserving was proposed by Linke Guo et al. [12], which applies users' attributes to find suitable friends and establishes social relationships with strangers via a multihop trust chain. Similarly, Squicciarini et al. [23] used game theory to model the privacy management for content sharing. This work can provide automatic access policy generation for users' profile information. In 2015, K. Samanthula et al. [13] utilized the concept of protecting the source privacy through anonymous message routing to recommend friends accurately and efficiently.

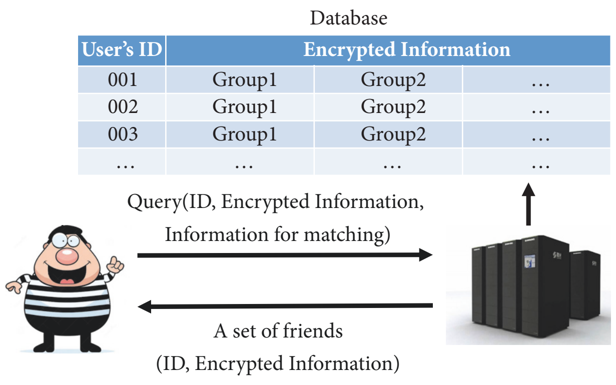
In 2018, a data sanitization strategy [24] was proposed which keeps the benefits brought by social network data, while sensitive latent information can still be protected. Even considering powerful adversaries with optimal inference attacks, the proposed strategy can still preserve both data benefits and social structure while guaranteeing optimal latent-data privacy [3]. To resist attacks, SmartMask, a context-based system-level privacy protection solution, was proposed, which was designed to learn users' privacy preferences under different contexts and provide a transparent privacy control for MSN users [25].

Furthermore, because of the importance of photos in friend-recommendation applications, several methods used for mainly protecting the images were proposed. A tool called iPrivacy (image privacy) was developed for releasing the burden from users on setting the privacy preferences when they share their images for special moments. It realizes photo protection by blurring the privacy-sensitive objects automatically [26]. Besides, a concept of changing the granularity of access control from the level of the photo to that of a user's personally identifiable information (PII) was proposed. In this work, it considers the face as the PII. When another user attempts to access a photo, the system determines which faces the user does not have the permission to view and presents the photo with the restricted faces blurred out. However, this mechanism can only be used for photo protection. The scenarios they can be applied on are limited [27].

Moreover, some existing ways of protecting privacy often introduce a third party [4]. All of the keys are stored in the third part. The information should be encrypted before being sent to the server and be decrypted while the server sends it back. However, in the solution of introducing a third party, the privacy information scattered in various service providers will be concentrated. We cannot guarantee the third party is able to protect those keys and the information.

3. Framework

The whole purpose of the proposed framework is to provide a user with the accurate social information according to its query without leaking any query information to the friend-recommendation application system. Query information is


Database
Query(ID, Encrypted Information,
Information for matching)
A set of friends
(ID, Encrypted Information)

User's ID	Encrypted Information		
001	Group1	Group2	…
002	Group1	Group2	…
003	Group1	Group2	…
…	…	…	…

Figure 1: The proposed privacy preservation framework for friend-recommendation applications.

User's ID		
Attributes		Personal Information
Group1	Age	Name
	location	
	…	Personal statement
Group2	Favorite movies	
	Favorite sports	Individual Signature
	…	
…	...	…

Figure 2: The information structure.

often sensitive because it is related to user's privacy. It always contains a user's personal information such as habits, hobbies, and preferences. Hence, our framework aims to realize zero-knowledge privacy protection. Here, *zero knowledge* implies the server knows nothing. Therefore, our aim is challenging, because on the one hand, the server needs information from the user to provide service, but on the other hand, such information needs to be kept secret to the server.

To realize the seemly ambivalent aims, we propose a privacy preservation framework for friend-recommendation application systems. The preservation is described in the following steps. The framework is shown in Figure 1.

(1) The friend-recommendation application server broadcasts three pieces of information to all users: (a) the attributes users can submit; (b) the attribute groups; (c) the part of keys used to decrypt users' information, each of which corresponds to one attribute group's value and hence different attribute groups' values have different keys. The reason why part of keys is related to users' attribute values is that this will make users who have the same attribute groups' values be able to decrypt the corresponding attribute values of the recommended friends.

(2) As required by the server, the user's information like name, age, hobbies, etc. are divided into several groups. For each group, the information is encrypted by homomorphic encryption algorithm. Then user's ID attached to all of the encrypted information will be sent to the server. The key used for encryption consists of two parts. The first part corresponds to the user's attribute values, which is obtained from the broadcast message sent from the server and the other is selected by the user freely. Hence, the key of each attribute group's value is different. Besides, our framework provides two matching patterns: Matching with Strict and Matching with Loose. For the Strict pattern, it requires that all the attribute groups of the friends recommended by the server should match those appointed by the user. As for the Loose pattern, it only requires that part of the attribute groups of the fiends recommended by the server matches those appointed by the user. The user needs to appoint the matching type and specific attribute groups. In the query information, we use 0 to represent the Strict pattern and 1 to represent the Loose pattern. The detailed content will be introduced in the Matching Algorithm section.

(3) Once receiving the information from a user, the server saves the information and finds friends who have the same attributes with the user using the matching algorithm based on homomorphic encryption algorithm in its database. For example, if a user sends his information: age twenties, love watching movies, the algorithm will find people who have nearly the same age and same preference in the database. It is worth noting that because all of the information in the server is encrypted, the user's privacy will be protected.

(4) The server returns the recommended friends' ID and information encrypted to the user. The user can select suitable people and get a connection with them using their ID. Moreover, the user can decrypt part of friends' information.(Through decryption, the user can find out which attributes are the same as those of the recommended friends.) This can be achieved because when a group of attributes of two users is the same, the corresponding keys of two users will be the same. These are related to our matching pattern, especially Loose pattern. How to deal with it will be introduced in the next section in detail.

At Step (1), before a user sends the query information to the server, its information needs to be encrypted first. Actually, the information of the user is divided into two types in our framework. The first one is attributes including hobbies, age, etc. The second type, such as name and personal statement, is merely the personal information. Because of the different characteristics of the information, we program them in different ways. It will be discussed in the Simulation section. The information structure of a user is depicted in Figure 2.

During the query process, there exists one potential privacy leakage issue. Specifically, If a hacker acting as a user asks for recommended friends, he will receive several users' information and know that their information is similar to that of him. In this way, he can acquire users' private information.

To solve this problem, we use users' locations as a special attribute [16] and divide other attributes into groups to reduce the probability of privacy attack from the malicious hackers. With location attributes, the server only recommends friends who are near to users. Grouping attributes make the user be able to only decrypt part of attributes of recommended friends. These methods enable users' privacy to be protected, not only when people want to look for friends, but also when they are recommended friends.

In detail, we separate the area into small equally sized grids whose vertex's longitude and latitude are both integers and transform user's coordinate into the nearest grid vertex. (The reason that the longitude and latitude of a grid's vertex are required to be integers is that it is easy for homomorphic encryption to deal with integers.) Now we introduce the following definition of location.

Definition 1 (a user's coordination). a user's coordination is that of a grid's vertex which is the nearest to the user's current location.

Definition 2 (district). The focal user's coordination is the center of the district which is a square whose length of the side is given. The length of the side of the district is an integral multiple of that of the grid.

In order to reduce the probability of information leakage, the proposed framework only recommends people whose district overlaps that of the user. This requires that the user who sends query should attach his location to the attributes.

For the sake of increasing the security of privacy further, we separate the attributes into several groups. Therefore, when recommending friends to the user, the attributes of two users do not need to be exactly the same, but only part of them. This approach makes sure that when users are recommended, only part of the information of the user will be known. This reduces the probability of leakage of user's privacy to a certain extent.

So far, the user can find friends who are suitable for him. In the whole process, the server operates on the cipher-text. This ensures that the user's private information will not be leaked. Even if a hacker invades the server, there is no chance that he can get any useful information because all the information is encrypted. Besides, even if the server knows the part of the keys corresponding to each attribute group's value combination, it is not easy for the server to decrypt the information of the user. This is because there are too many combinations of the attribute group's value, so it is hard to try all the possible keys to decrypt the information of users. We will further discuss how it is difficult for the server to attack the privacy of users in the Simulation section.

4. Algorithm

In order to realize zero-knowledge privacy protection, our framework adopts algorithms based on homomorphic encryption [28–30]. Homomorphic Encryption allows complex mathematical operations on cipher-texts, generating an encrypted result which matches the result which is operated on the plaint-text and then decrypted [29]. The purpose of homomorphic encryption is to realize a more secure method for data processing. In the following, we detail our algorithms based on homomorphic encryption.

```
Input:
  $M$: all the attributes groups( $M = M_1, M_2....M_k$ );
  $M_i$: the $i$ attributes group;
  $I$: personal information ( $I = I_1, I_2....I_k$ );
  $I_i$: $i$ personal information version;
  $P_i$: the key of $i$ attributes group;
  $Q_i$: the key of $i$ attributes group;
  $R_i$: the key of $i$ attributes group;
  $K$: the number of attributes groups;
Output:
  $C([M])$: all the encrypted attributes (
  $M = M_1, M_2....M_k$ );
  $C([I])$: all the encrypted personal information (
  $I = I_1, I_2....I_k$ );
(1) for $i = 1; i \leq K; i++$ do
(2)     $N_i \longleftarrow P_i \times Q_i$;
(3)     if $R_i \neq 0$ then
(4)         $C([M_i]) \longleftarrow (M_i + P_i \times R_i) \bmod N_i$;
(5)         $C([I_i]) \longleftarrow (I_i + P_i \times R_i) \bmod N_i$;
(6)     end
(7) end
(8) return $C([M]), C([I])$;
```

Algorithm 1: Encryption.

4.1. Encryption. To preserve the privacy of users, the query needs to be encrypted by homomorphic encryption algorithm, shown in Algorithm 1. In order to guarantee the safety of the user's information, as mentioned in the above section, the key used to encrypt information consists of two parts: one corresponds to the attribute groups' value, and the other is selected by the user randomly. Hence, the key of each attribute group is different. Let M_i represent the user's attribute group i, I be the user's personal information, I_i be the i personal information, and K represent the number of attribute groups. The corresponding encryption process is depicted as follows:

For every attribute group i, according to the broadcast information, the client will select two safe large prime numbers P_i and Q_i according to the attribute values of the user. If the users' attribute values are the same, the generated P_i and Q_i are the same. This will make sure that if two users are matched, they can encrypt at least part of the information of each other. The client generate a product $N_i = P_i * Q_i$.

Using the encryption algorithm, the value of each attribute group i is encrypted to the cipher-text $C([M_i])$. $C([M_i]) = (M_i + P_i \times R_i) \bmod N_i$, where R_i is selected by the user randomly to guarantee the information security.

The user's personal information I is encrypted by the key of each attribute groups i apart into several

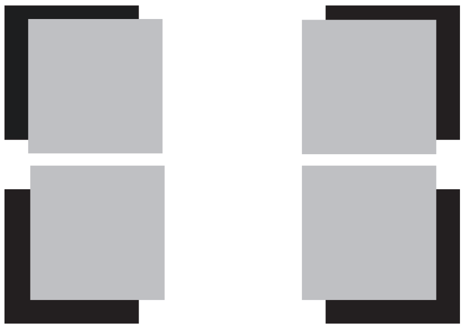


FIGURE 3: The location matching.

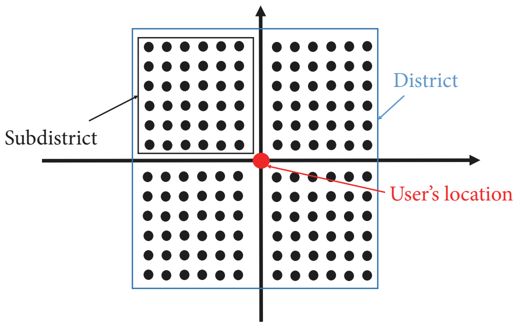


FIGURE 4: The location structure.

```
    $k$ is the number of queue which saves the vertexes.
    $m^2$ is the number of vertexes in the subdistrict.
    **Input:**
    $La$: $user_a$'s information;
    $Lb$: $user_b$'s information;
    **Output:** true or false
(1) $result \longleftarrow 0$;
(2) $loop$: **for** $k = 1; k \le 4; k++$ **do**
(3)     **for** $i = 1; i \le m; i++$ **do**
(4)         **for** $j = 1; j \le m; j++$ **do**
(5)             **if** $La_k i == Lb_k j$ **then**
(6)                 $result++$;
(7)                 $break\ loop$;
(8)             **end**
(9)         **end**
(10)    **end**
(11) **end**
(12) **if** $result == 4$ **then**
(13)     return $true$;
(14) **end**
(15) **else**
(16)     return $false$
(17) **end**
```

ALGORITHM 2: Location matching.

versions $C([I_i]) = (I_i + P_i \times R_i) \bmod N_i$. In this way, it can make sure that even if only one attribute group of the user is the same as that of the recommended friends, the friends' personal information can still be decrypted by the key of this attribute group. Therefore, the users can know each other better.

4.2. Matching Algorithm. When receiving the user's ID and the information encrypted, the server runs the matching algorithm to find suitable recommended people for the user in the database. The matching algorithm mainly includes two parts: location matching and attribute matching.

4.2.1. Location Matching. The purpose of location matching is to find people whose district overlaps that of the user. As shown in Figure 6, the grey square represents $user_1's$ district, and the black square represents $user_2's$ district. If these two districts overlap, then it must look like one of the situations in Figure 3. The overlapping can appear in the upper left, upper right, lower left, and lower right in the square. To simplify the matching algorithm, we see the user's location as the center of a rectangular coordinate system and divide the district into four parts based on the quadrant of the rectangular coordinate system. It is shown in Figure 4. Each vertex in the subdistrict is evenly distributed. The basic idea of the proposed location matching algorithm is that we compare two users' districts. If each subdistrict has the same vertices with the corresponding one, then we consider these two users' districts overlap.(Because homomorphic encryption algorithm can only judge whether the information is equal, we adopt the local discretization method for the location matching. The more vertexes in the district are, the more precise the matching algorithm is, and vice versa.)

Algorithm 2 judges whether two users' districts overlap: La which includes four queues holds $user_a$'s district information and Lb holds $user_b$'s district information. The algorithm is going to check four queues with their corresponding one (Lines (2)). For each queue, all the vertexes will be compared with those in the corresponding one (Lines (3)-(4)). When there are vertexes matched, then the algorithm tags this queue as matched (Lines (5)-(8)). If all the queues are matched with their corresponding one, we consider that these two users' districts are matched and return true (Lines (12)-(14)). Otherwise, we return false (Lines (15)-(17)).

4.2.2. Attributes Matching. When the user makes a query, the server will receive query information which is in the form of cipher-text. We use $C([M])$ to represent it. Because the server has stored the information of other users, the matching algorithm will compare each record in the database with the query information to find the suitable recommended people. The operation result is computed according to the following algorithm, where $C([Ma_i])$ represents the i attribute group in the form of cipher-text of the $user_a$ and Q_i, N_i represent the i attribute group's parameters. Similarly, $C([Mb_i])$ represents the i attribute group of $user_b$ in the database in the form cipher-text. K represents the number of attribute groups. If $res_i = 0$, the i attribute group of $user_a$ matches that of $user_b$ in the database. Otherwise, they are mismatched, and the server will compare other user's attribute groups with the cipher-text of the user.

```
$K$: the number of attribute groups.
Input:
$C([Ma_i])$: the $i$ attribute group of $user_a$ in the form
of cipher-text( $i = 1, 2, 3...k$ );
$C([Mb_j])$: the $i$ attributes group of $user_b$ in the form
of cipher-text( $i = 1, 2, 3...k$ );
$Q_i$, $N_i$: the $i$ attribute group's parameters set by the
$user_a$;
Output: $res$: the matching result (
        $res = res_1, res_2....res_k$ )
(1) if $Q_i \neq 0$ AND $N_i \neq 0$ then
(2)     for $i = 1; i \leq K; i++$ do
(3)         $res_i \longleftarrow ((C([Ma_i]) - C([Mb_j])) \times Q_i)$
            $\bmod N_i$;
(4)     end
(5) end
(6) return res;
```

Algorithm 3: Attributes matching.

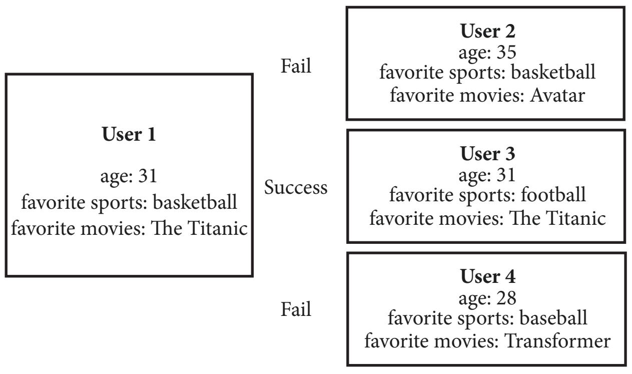
Algorithm 3 shows the attributes matching process. The explanation of the detailed analysis is as follows: We derive the calculation formula in the Algorithm 3. It can get the following form. Ma_i and Mb_i represent the i attributes groups of $user_a$ and $user_b$ in the form of plain-text separately. Ra_i represents the part of the key of the i attribute group of $user_a$ and Rb_i represents the part of the key of $user_b$.

$$\begin{aligned} res_i &= \left(\left(C\left([Ma_i]\right) - C\left(\left[Mb_j\right]\right)\right) \times Q_i\right) \bmod N_i \\ &= \left(\left(\left(C\left([Ma_i]\right) + P_i \times Ra_i\right)\right.\right. \\ &\quad \left.\left. - \left(C\left(\left[Mb_j\right]\right) + P_i \times Rb_i\right)\right) \times Q_i\right) \bmod N_i \\ &= \left(\left(C\left([Ma_i]\right) - C\left([Mb_i]\right)\right) \times Q_i + \left(Ra_i - Rb_i\right) \times P_i\right. \\ &\quad \left. \times Q_i\right) \bmod N_i \end{aligned} \tag{1}$$

Because $N_i = P_i \times Q_i$, then $(Ra_i - Rb_i) \times P_i \times Q_i \bmod N_i = 0$; if $Ma_i == Mb_i$, then $((C([Ma_i]) - C([Mb_i])) \times Q_i) \bmod N_i = 0$. Otherwise, $((C([Ma_i]) - C([Ma_i])) \times Q_i) \bmod N_i > 0$. Therefore, if $Ma_i == Mb_i$, then res=0; otherwise, $res > 0$.

Considering matching, the server could request that all of the attributes be matched, but generally we do not need this. Objectively, it is tough to find two people whose attributes are the same. Subjectively, the purpose of dating is not to find a mirror image of the user, but to find suitable friends. Therefore, when it comes to matching algorithm, we mainly consider that parts of the attributes are the same. The partial matching is divided into two parts: Strict Matching and Loose Matching. We will introduce them in detail.

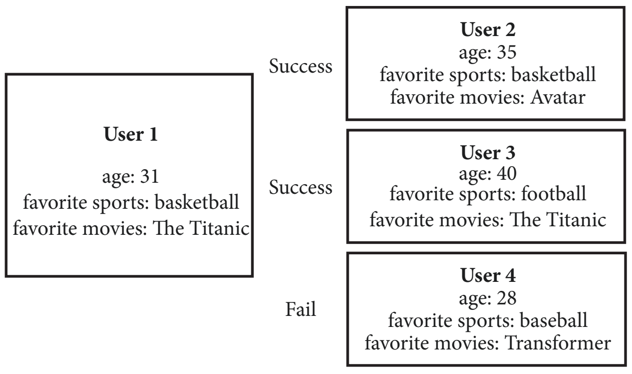
(1) Strict Matching: When the user is sending his query, he must have specific demands looking for friends. The user needs to appoint some attribute groups. The Strict Matching Attributes algorithm should make sure that these attribute groups are matched. In this way, although the server saves a large number of attributes of users, only part of the attributes will be used in the matching algorithm. It not only meets the needs of the users but also saves the operation cost to a great extent, which makes the matching algorithm more efficient. For example, as shown in Figure 5, let one user's attributes be {age: 31, favorite sport: basketball, favorite movie: The Titanic}. Each attribute is a group. Then, he points out that the friend he wants should like the same movie and have the same age. When the server responds, the recommended friends' favorite sport could be different, but they must love to watch The Titanic and have the same age.



User 1
age: 31
favorite sports: basketball
favorite movies: The Titanic
Fail
Success
Fail
User 2
age: 35
favorite sports: basketball
favorite movies: Avatar
User 3
age: 31
favorite sports: football
favorite movies: The Titanic
User 4
age: 28
favorite sports: baseball
favorite movies: Transformer

Figure 5: Strict Matching.

(2) Loose Matching: As for the Loose Matching algorithm, it only needs to make sure that part of the attribute groups which the user appoints is matched. If the user appoints N attributes and he gives a number M ($M \leq N$), the server only needs to check N attribute groups and makes sure whether there are matched M attribute groups. It does not need all the N attribute groups to be matched. It not only does not care whether all of the attribute groups are matched but also does not pay attention to which attribute groups appointed are matched. In other words, the Strict Matching is a particular case in Loose Matching.



User 1
age: 31
favorite sports: basketball
favorite movies: The Titanic
Success
Success
Fail
User 2
age: 35
favorite sports: basketball
favorite movies: Avatar
User 3
age: 40
favorite sports: football
favorite movies: The Titanic
User 4
age: 28
favorite sports: baseball
favorite movies: Transformer

Figure 6: Loose Matching.

Using the example above, shown in Figure 6, let one user's attributes be {age: 31, favorite sport: basketball, favorite movie: The Titanic}. Each attribute becomes a group. Then, he appoints movie and age as the matching condition and gives a number $M = 1$. When the server respondes, the recommended people's age and favorite movie could be different, but at least one of them is the same.

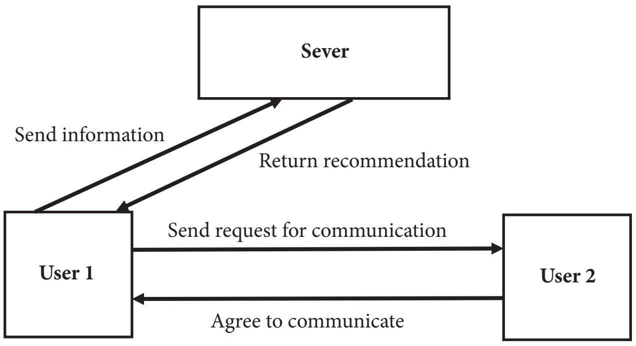


Figure 7: The decryption process.

4.3. Decryption. Having said that, we should divide the situations into different parts. First, it is not necessary to present the recommended friends' information to the users. In order to make sure that the user is not malicious dating, we can hand the right of showing the information to the recommended people. All of the information of any user who is searching friends is only used to run matching algorithm in the server. The information will not be sent to any user. The user could only get an "ID" of the recommended friends and an Internet connection which is used for chatting. Therefore, if the user sincerely wants to make friends, the server will tell him that this is the suitable person, but the next things should depend on him. In this circumstance, the decryption is useless. The process is shown in Figure 7.

Second, sometimes it is necessary to give users some information about the recommended people. This will help them to know each other and make friends more quickly. In this case, the decryption plays an important role. The user will receive the cipher-text packet, decrypt it using decryption algorithm, and get the plain-text. In some situations, the matching algorithm does not need all of the attributes to be matched.(Some attribute groups can be different.) Therefore, the decryption is only used for those attribute groups which are matched. The query user will send a request to the recommended people asking for information. If the recommended people agree on the query user's request, the query user can use the homomorphic decryption to obtain plain-text of the information. The decryption algorithm is shown in Algorithm 4.

In Algorithm 4, K is the number of attribute groups. M_i represents the i attribute group of recommended friends in the form of plain-text, and M represents all the attribute groups. P_i is the key of the i attribute group of the user. res is the matching result.

```
K: the number of attribute groups.
M_i: the i attribute group of recommended friends in
  the form of plain-text( $i = 1, 2, 3...k$ )
Input:
$C([M_i])$: the i attribute group of recommended
friends in the form of cipher-text( $i = 1, 2, 3...k$ );
$P_i$: the i attribute group's key of user;
$res$: the matching result ( $res = res_1, res_2....res_k$ );
Output: $M$: all the attribute groups in the form of
        plain-text( $M = M_1, M_2....M_k$ );
(1) if $P_i \neq 0$ then
(2)     for $i = 1; i \leq K; i++$ do
(3)         if $res_i == 0$ then
(4)             $M_i \longleftarrow C([M_i]) \bmod P_i$
(5)         end
(6)     end
(7) end
(8) return $M$;
```

Algorithm 4: Decryption.

5. Defense for Specific Attack

Strictly, none of the encryption algorithms are safe mathematically. Certainly, the server or the hacker can still unremittingly try every possible key to decrypt all the information. Considering the cost of fee and time, it is unpractical in the current technology. However, they could act as a user to match people who have registered the same information in the server. In other words, as a hacker, he does not receive the whole information of the user, but some attributes. For example, if a hacker sends some attributes {age: 25, favorite sport: football, favorite movie: The Titanic} to the server. Then he points out that the friend he wants should have the same age and favorite sport. When the server respondes, there are some recommended friends sent back, and the hacker will know these people's age and favorite sport. In this situation, we adopt Obscure Matching based on dividing attributes into several groups, as well as district limitation, as we mentioned briefly in Section 4. In this section, we introduce our defense to this attack in detail.

5.1. Loose Matching. Dividing attributes into several groups makes room for partial matching, aiming to reduce the probability of information leakage. Based on the attribute groups, we propose Loose Matching further. It only requires that part of the attribute groups of the friends recommended by the server matches those appointed by the user. In this way, the attributes groups which are not matched by the user will not be known by the user. It not only meets the needs of the users but also reduces the probability of information leakage.

Using the example above, let the hacker's attributes be {age: 25, favorite sport: basketball, favorite movie: The Titanic}. Each attribute becomes a group. Then he points out that the recommended people should like the same sport and have the same age. When the server respondes, the recommended people may have the same age or like the same sport, but the hacker does not know which attribute is the same.

5.2. District Limitation. As we mentioned above, we talk about the location attribute. It also decreases the probability of information leakage. With the district limitation, the server can not recommend all the people who have the same attributes. In this way, we can not only reduce the burden

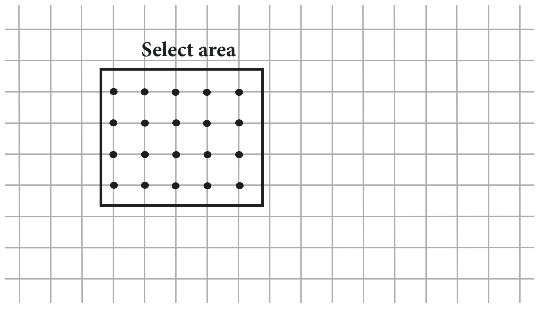
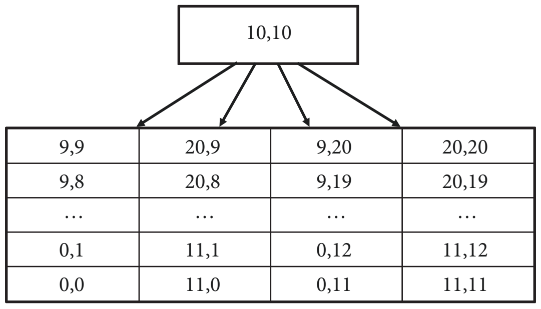


Figure 8: The district limitation.



9,9	20,9	9,20	20,20
9,8	20,8	9,19	20,19
⋯	⋯	⋯	⋯
0,1	11,1	0,12	11,12
0,0	11,0	0,11	11,11

Figure 9: The location structure.

of the whole network but also prevent some malicious users to acquire other user's information. This also reduces the possibility for the hacker to pretend to be a user to acquire information. Below we give further analysis.

Suppose there are M hackers served by the server and there are K location vertexes covered by the database. The number of location vertexes near to the user is L. As shown in Figure 8, it is a select area. From the perspective of a user, without the district limitation, the possibility that the user is recommended to the hacker is 1. However, with the district limitation, the number of hackers near to the user is $(M \times L)/K$. Therefore, the possibility that the user is recommended to the hacker will be reduced to L/K. The more the location vertexes are, the smaller the possibility is.

For example, according to the longitude and latitude, there are 360×180 vertexes which are the integer coordinates in the world. Let us presume that there are 10000 hackers in the world. Based on the regulation of location attributes, only the people near to the user will be recommended. The number of the near coordinates which could be recommended to the user is 100. Therefore, with the district limitation, the possibility that the user is recommended to the hacker is $100/64800 = 0.0015$. The number of the hacker who can acquire the user's information is $(100 * 10000)/64800 = 1500$. It is clear that through the district limitation, the possibility of user information leakage has largely decreased. As for the users who are near to each other, we should believe that because they are close, they intend to make friends with each other. The information that they get is necessary.

6. Simulation

We build a project which is based on the C/S system to realize our framework. The client which resides in the user's machine is used to provide a service of recommending friends.

As mentioned before, the client mainly realizes dividing the attributes into groups and encrypting the user's information. When receiving recommended friends from the server, it also decrypts attributes which are matched. We code all of this with Java. As for user's information, the attributes are programmed directly as Class (Java). These are the core of the user's information which will be used frequently. Then, we build a pointer in the Class which points to the personal information. In this way, the runtime of the matching algorithm will be accelerated, and the storage space will be saved.

The function of the server is to run the location and attributes matching algorithms. For the attributes matching, we need to covert the texts into numbers before encryption and transform numbers into texts after receiving the recommended friends' information. Actually, these two conversions are the same essentially. Taking the conversion from numbers to texts, for example, we correspond each number to a letter according to the ASCLL. As for some letters corresponding to a number which only has two digits, we add 9 to the hundreds place. Hence, each letter corresponds to a three-digit number. In this way, each big number corresponds to a word. Let the big number '109117115105999' to be an example. The corresponding letters of each three-digit number are 109-m, 117-u, 115-s, 105-i, and 999-c. Therefore, the corresponding word of this big number is 'music'. For the location matching, as mentioned above, the server views the user's location as a center to build a district. Each district consists of four subdistricts. The coordinates of vertexes which compose different subdistricts are saved in different queues. When using the matching algorithm, we compare the four queues with their corresponding one separately. The detailed process of generating the district according to the user's coordinates is as follows:

> We view the user's location as the center of the plane rectangular coordinate system. Take the first quadrant for example. We add from 1 to 10 to the longitude and the latitude of the user's current location to create 100 new vertexes to build a subdistrict. We do the same operation on the other quadrants.
>
> We use four queues to save the information of four quadrants. The vertex's coordinates from the same quadrants will be saved in the same queues. Figure 9 shows, when the user's coordinates are $(10, 10)$, how is the data of the subdistrict saved in the four queues.
>
> The client will encrypt all of the coordinates of vertex. At the same time, the order of the vertexes in each queue should be disorganized to make sure the server could not find the center vertex in the queues.

The server is also coded with Java. We built the server with many sockets, such as Register Socket, Match Socket, Modify Socket, etc. As the most crucial socket, the Match

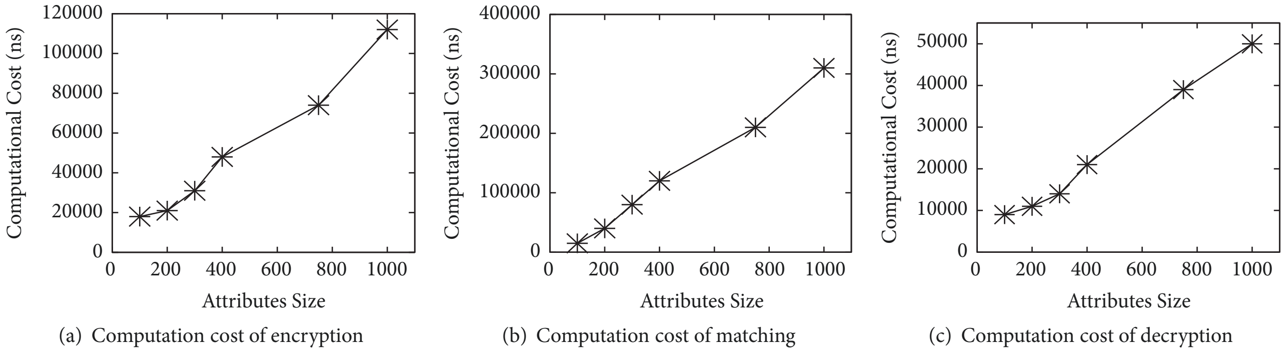


Figure 10: The speed of encryption, matching, and decryption.

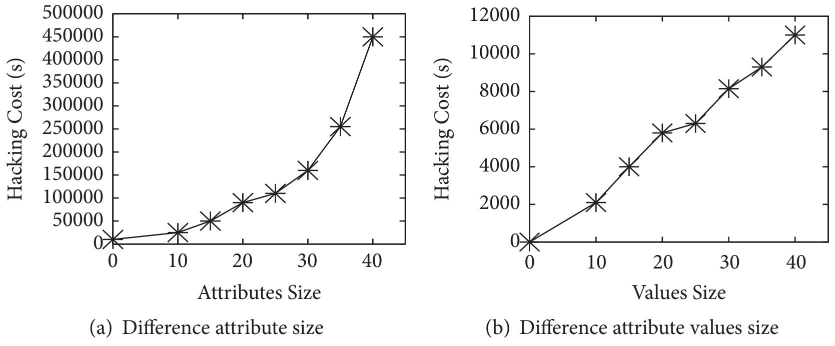


Figure 11: Possibility of decryption.

Socket receives the users' ID and attributes and sends the information in the server layers by layers to the database layer. Then the server will open the database, retrieve information, and run the matching algorithm [31]. After that, the server packs all of the suitable recommended friends' information and sends it back to the users. Finally, the client uses this information like users' ID to make a connection with the potential friends.

The experiments are run on two Windows machines with 2.4GHz CPU, 8 GB memory, 1TB 7200 RPM hard-disk. One is used for the server. The other runs the client.

In our simulation, we focus on several important metrics. The speeds of encryption, decryption, and matching as well as the possibility of decrypting the information by the server or the hacker.

6.1. The Speed of Encryption, Matching, and Decryption. Figure 10, respectively , illustrates how the time of encryption, matching, and decryption change with different attribute numbers M. Here, $M = 100, 200, 300, 500, 750, 1000$ and all the information is encrypted by the same two prime numbers, i.e., $P = 19961993$, $Q = 2013265921$. It is clear that the encryption, matching, and decryption costs are almost linearly proportional to M. As the the attribute size increases, the time needed to wait for sending information will be increased.

6.2. Possibility of Decryption. The possibility of the server decrypting the information is important since we need to make sure that, with the current technology, there is low chance to leak the information to the user. Therefore, we try to decrypt the cipher-text and record the cost from a hacker's angle. (We assume that there are hackers or the server does an inside job to act as a hacker.) We put M attributes in an attribute group, where M=10, 15, 20, 25, 30, 35, and 40, and each attribute has 5 values. For example, as for 30 attributes in one group, we use 300 big prime numbers divided into 150 groups to encrypt attribute groups. For each different combination of the values to one attribute group, we use different groups of prime numbers for decryption. If the server wants to decrypt the information, it will need to try 75 times on average (searching in disordered arrangement [32]). It is clear that the time of decryption by the server is almost squarely proportional to the numbers of attributes. As the number of attributes increases, the difficulty of the server decrypting the information will be increased.

Figure 11(a) shows the cost of hacking when the number of attribute values is invariant and the number of attributes changes. In this part, we test the hacking cost when the number of attribute values changes and the number of attributes is invalid. We take 10 attributes in an attribute group, and each attribute has 10, 15, 20, 25, 30, 35, and 40 values, respectively. For example, as for 30 values, we use 300 big prime numbers divided into 150 groups to encrypt attribute groups. For each different combination of the values to one attribute group, we use different groups of prime numbers for encryption. If the server wants to decrypt the information, it needs to try 75 times on average (searching in disordered arrangement [32]). Figure 11(b) shows the time the server needs to decrypt the information. It is obvious that the time of decryption by the server is almost linearly proportional to the numbers of values of the attributes. As the number of values of attributes increases, the difficulty of the server decrypting the information will be increased. We can conclude that the chances of decryption by the server will be low if the number of attributes and the number of values of attributes are big enough.

7. Conclusion

In order to better protect the user's personal information on the friend-recommendation applications, we put forward the privacy protection mechanism based on zero knowledge. The client encrypts the information using homomorphic encryption and sends the cipher-text to the server. Then the server helps the user to find people who have the same attributes based on homomorphic encryption. In the whole process, the user does not need to worry about whether his information is in danger. Once his information leaves the client, it is in the form of cipher-text. Meanwhile, the server can still finish its work smoothly. We propose a method of district limitation for matching operation which can decrease the possibility of information leakage, and grouping attributes enhance the security of our framework. Finally, the defenses of grouping attributes and location limitation largely reduces the risk of being leaked by the hacker. Besides, the Loose Matching also provides much protection. This mechanism can realize that although a friend-recommendation system server does not know any users' information, it can help user match friends who are in accordance with the same attributes. This scheme not only has reference and practicality for practical application, but also has excellent scalability. It provides a strong basis for the future perfection and enhancement.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

An early version of the manuscript has been presented as conference paper in 2017 International Conference on Identification, Information and Knowledge in the Internet of Things [33].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work has been supported by the National Natural Science Foundation of China (No. 61472044, No. 61772080, No. 61472403) and the Natural Science Foundation of Beijing (No. 4182062).

References

[1] X. Zhang, Z. Yang, Z. Zhou, H. Cai, L. Chen, and X. Li, "Free market of crowdsourcing: Incentive mechanism design for mobile sensing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3190–3200, 2014.

[2] X. Zhang, Z. Yang, W. Sun et al., "Incentives for mobile crowd sensing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 54–67, 2016.

[3] Z. He, Z. Cai, and J. Yu, "Latent-data privacy preserving with customized data utility for social network data," *IEEE Transactions on Vehicular Technology*, vol. PP, no. 99, pp. 1-1, 2017.

[4] J. Wen-guang and S. Yu-qing, "Personalized privacy protection for third-party service platform," *Journal of Lanzhou University, Natural Sciences*, vol. 48, no. 4, pp. 85–90, 2012.

[5] D. Chen and H. Zhao, "Data security and privacy protection issues in cloud computing," in *Proceedings of the Proceeding of the International Conference on Computer Science and Electronics Engineering (ICCSEE '12)*, vol. 1, pp. 647–651, Hangzhou, China, March 2012.

[6] A. Vapen, N. Carlsson, A. Mahanti, and N. Shahmehri, "Information sharing and user privacy in the third-party identity management landscape," *IFIP Advances in Information and Communication Technology*, vol. 455, pp. 174–188, 2015.

[7] K. Vu, R. Zheng, and J. Gao, "Efficient algorithms for K-anonymous location privacy in participatory sensing," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '12)*, pp. 2399–2407, Orlando, Fla, USA, March 2012.

[8] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[9] F. Yu, M. Chen, B. Yu, W. Li, L. Ma, and H. Gao, *Privacy Preservation Based on Clustering Perturbation Algorithm for Social Network*, Springer Science and Business Media, 2017.

[10] C. Zhipeng and Z. Xu, *A Private and Efficient Mechanism for Data Uploading in Smart Cyber-Physical Systems*, Transactions on Network Science and Engineering (TNSE).

[11] Y. Liang, Z. Cai, J. Yu, Q. Han, and Y. Li, "Deep learning based inference of private information using embedded sensors in smart devices," *IEEE Network*, vol. 32, no. 4, pp. 8–14, 2018.

[12] L. Guo, C. Zhang, and Y. Fang, "A Trust-Based Privacy-Preserving Friend Recommendation Scheme for Online Social Networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 413–427, 2015.

[13] B. K. Samanthula, L. Cen, W. Jiang, and L. Si, "Privacy-preserving and efficient friend recommendation in online social networks," *Transactions on Data Privacy*, vol. 8, no. 2, pp. 141–171, 2015.

[14] X. Zheng, Z. Cai, and Y. Li, "Data Linkage in Smart Internet of Things Systems: A Consideration from a Privacy Perspective," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 55–61, 2018.

[15] X. Zheng, Z. Cai, G. Luo, L. Tian, and X. Bai, "Privacy-preserved community discovery in online social networks," *Future Generation Computer Systems*, 2018.

[16] X. Zheng, Z. Cai, J. Li, and H. Gao, "Location-privacy-aware review publication mechanism for local business service systems," in *Proceedings of the IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9, Atlanta, GA, USA, May 2017.

[17] L. Zhang, Z. Cai, and X. Wang, "FakeMask: A Novel Privacy Preserving Approach for Smartphones," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 335–348, 2016.

[18] Z. He, Z. Cai, and X. Wang, "Modeling propagation dynamics and developing optimized countermeasures for rumor spreading in online social networks," in *Proceedings of the 35th IEEE International Conference on Distributed Computing Systems (ICDCS '15)*, pp. 205–214, July 2015.

[19] X. Qin, Y. Luo, N. Tang, and G. Li, "DeepEye: An automatic big data visualization framework," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 75–82, 2018.

[20] L. Shi, Y. Wu, L. Liu, X. Sun, and L. Jiang, "Event detection and identification of influential spreaders in social media data streams," *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 34–46, 2018.

[21] H. Zang and J. Bolot, "Anonymization of location data does not work: a large-scale measurement study," in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking (MobiCom '11)*, pp. 145–156, ACM, September 2011.

[22] R. Lu, X. Lin, Z. Shi, and J. Shao, "PLAM: A privacy-preserving framework for local-area mobile social networks," in *Proceedings of the 33rd IEEE Conference on Computer Communications, IEEE INFOCOM 2014*, pp. 763–771, Canada, May 2014.

[23] A. C. Squicciarini, F. Paci, and S. Sundareswaran, "PriMa: A comprehensive approach to privacy protection in social network sites," *Annals of Telecommunications-Annales des Télécommunications*, vol. 69, no. 1-2, pp. 21–36, 2014.

[24] Z. Cai, Z. He, X. Guan, and Y. Li, "Collective data-sanitization for preventing sensitive information inference attacks in social networks," *IEEE Transactions on Dependable and Secure Computing*, 2016.

[25] H. Li, H. Zhu, S. Du, X. Liang, and X. Shen, "Privacy leakage of location sharing in mobile social networks: attacks and defense," *IEEE Transactions on Dependable and Secure Computing*, no. 99, 2016.

[26] J. Yu, B. Zhang, Z. Kuang, D. Lin, and J. Fan, "IPrivacy: image privacy protection by identifying sensitive objects via deep multi-task learning," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1005–1016, 2017.

[27] P. Ilia, I. Polakis, E. Athanasopoulos, F. Maggi, and S. Ioannidis, "Face/off: preventing privacy leakage from photos in social networks," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, (CCS '15)*, pp. 781–792, October 2015.

[28] Z. Khan, "Qbuasi-Linear Time Fully Homomorphic Public Key Encryption Algorithm (ZK111)," *Journal of Theoretical Physics and Cryptography*, vol. 1, no. 1, pp. 14–17, 2012.

[29] Y. Yang, S. Zhang, J. Yang, J. Li, and Z. Li, "Targeted fully homomorphic encryption based on a double decryption algorithm for polynomials," *Tsinghua Science and Technology*, vol. 19, no. 5, pp. 478–485, 2014.

[30] K. Hariss, H. Noura, and A. E. Samhat, "Fully Enhanced Homomorphic Encryption algorithm of MORE approach for real world applications," *Journal of Information Security and Applications*, vol. 34, pp. 233–242, 2017.

[31] N. Ogura, G. Yamamoto, T. Kobayashi, and S. Uchiyama, "An improvement of key generation algorithm for Gentry's homomorphic encryption scheme from ideal lattices," *Journal of Math-for-Industry (JMI)*, vol. 3B, pp. 99–106, 2011.

[32] M. Li, J. Li, and C. Huang, *A Credible Cloud Storage platform based on Homomorphic Encryption*, Beijing University of Posts and Telecommunication, 2012.

[33] W. Wang and S. Wang, "Privacy Preservation for Dating Applications," *Procedia Computer Science*, vol. 129, pp. 263–269, 2018.