Cryptographic Protocols for Enforcing Relationship-based Access Control Policies

Jun Pang*†, Yang Zhang†

*Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg
†Faculty of Science, Technology and Communication, University of Luxembourg

***Abstract*—Relationship-based access control schemes have been studied to protect users' privacy in online social networks. In this paper, we propose cryptographic protocols for decentralized social networks to enforce relationship-based access control polices, i.e., k-common friends and k-depth. Our protocols are mainly built on pairing-based cryptosystems. We prove their security under the honest but curious adversary model, and we analyze their computation and communication complexities. Furthermore, we evaluate their efficiency through simulations on a real social network dataset.**

I. Introduction

Online social networks (OSNs) have gained a huge success during the past decade. Nowadays, using OSNs service almost becomes an indispensable part in people's daily lives. A user in OSNs can specify his profile, articulate his social relationships, share his life moments, etc. With more and more personal information appearing online, users' privacy has become an essential problem since, as in most of the times users don't want to share their lives with everybody. Access control is one of the most straightforward but useful ways to address this challenge. In recent years, industries have adopted relationship-based access control schemes for OSNs, for example, Facebook and LinkedIn. In such schemes, whether a user can view another user's information depends on the relationship between these two users. For example, a user in Facebook can decide which users can view his photos, based on their social depth to himself – being his friends, friends of friends, etc. However, social relationships as well as interactions among people are very complicated, they cannot be fully represented by "friends" and "friends of friends". For example, if two strangers share a lot of common friends, then the chance that they trust each other is high [1]. Thus, it is more probable that they want to interact with each other. On the other hand, if these two users only have one friend in common, even though their relationship is "friend of friend", the chance for them to communicate is much lower than the former case. Situations like this happen often but the current access control schemes in OSNs do not support them. Therefore, it is necessary to have more fine-grained access control schemes.

Several fine-grained relationship-based access control polices have been proposed by research communities [2], [3], [4]. These policies are necessary as they enable users to have more precise and strict control on who can access their personal information or resources. On the other hand, such policies are quite flexible and give users possibilities to regulate access control based on relationships and the topology of the underlying social graph. For instance, a user can define a policy allowing the users who have at least a number of common friend with him to review his photos. The user can also specify a policy that only his friends and friends of his family members can view his photos. Despite of their expressiveness and flexibility, implementing these policies normally requires many computing resources which are usually the bottleneck even for big companies, like Facebook and Google. Decentralized social networks (e.g., [5], [6]) have been proposed in the literature as an ideal solution to address the problem. In decentralized social networks, users can control their own data and enforce access control polices with their personal devices instead of putting the burden on the central operators' shoulder [7]. In this way, users can get rid of the central operators and implementing fine-grained relationship-based access control polices will then only involve social network users. In recent years, developing cryptographic protocols for enforcing fine-grained access control polices in decentralized social networks has been an active research area (see more discussions in Section II).

Contributions. In this paper, we propose cryptographic protocols to implement two fine-grained access control polices, i.e., 'k-common friends' and 'k-depth' as proposed in [4], for decentralized social networks. Our main contributions in this paper are summarized as follows.

- We propose the first protocol to enforce k-common friends policies. Our construction is based on paring-based cryptosystems (PBC) and private set intersection protocols. Security analysis shows that our protocol is secure under the honest but curious adversary model.
- Our protocol for k-depth policies is based on PBC as well. We have proved that our protocol is secure under the honest but curious adversary model. Compared to existing protocols, our protocol achieves a better security level. Our k-depth protocol can be extended to support multi-relationship social networks.
- For our protocols, we perform a detailed analysis of their efficiency and conduct an empirical evaluation of their performance with a real-life social network dataset. The results show that our protocols are quite practical.

Organization. After presenting related work in Section II, we introduce some preliminaries in Section III. The OSN model as well as access control polices are presented in Section IV. Our protocols for enforcing two relationship-based access control policies are described in Section V and Section VI together with corresponding security proofs, respectively. The theoretical as well as empirical evaluation of our protocols are presented in Section VII. Section VIII concludes the paper with some future work.

II. Related Work

Access control models. Carminati et al. [8] proposed the first relationship-based access control model where polices are based on three regulations including relationship type, depth and trust value. In [3], a fine-grained model based on semantic web tools is introduced, where they proposed 'admin' and 'filtering' polices. Fong et al. introduced an access control scheme which supports Facebook-style social networks [4]. Specifically, they proposed a few fine-grained access control polices including the topology-based ones that we will focus on in this paper. In [9], [10], [11], modal (hybrid) logics are exploited to express access control polices. Recently, Pang and Zhang [12] extended their work by defining a new OSN model containing users and their relationships as well as public information. Based on this new model, they introduced a new hybrid logic for formulating access control policies. Cheng et al. [13] presented a social network model where resources are also treated as nodes. With this model, access control polices, related to not only user-to-user but also user-to-resource and resource-to-resource relationships, are supported.

Security protocols. Besides access control models, security protocols for enforcing access control polices in OSNs have also been studied. Carminati et al. introduced several solutions [14], [15], [16]. For instance, in [16], a homomorphic encryption scheme is used to compute aggregated information of the path (trust level and relationship type), which minimizes the loss of sensitive information. In [17], Mezzour et al. proposed an interesting solution for path discovery where the protocol contains two stages – in the first stage, each user floods tokens in the social network, while in the second stage a private set intersection protocol is executed for finding paths. Backes et al. [18] defined a security API for distributed social networks, where cryptographic techniques such as pseudonyms, digital signatures and zero-knowledge proofs are exploited to help user to establish and prove the existence of friendships with others. In [19], a key management scheme was proposed under which only users who are within a certain distance to the owner are able to derive keys to decrypt the encrypted resources. A comparison between our protocols and two existing protocols is presented in Section VII.

III. Preliminaries

A. Cryptographic Building Blocks

Bilinear map. Let $G_1 = \langle g \rangle$ and G_2 be two multiplicative groups of the same prime order p. An efficient computable map $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear map if the following properties hold:

Bilinearity: $\forall a, b \in \mathbb{Z}_p^*,\ e(g^a, g^b) = e(g, g)^{ab}$
Non-Degeneracy: $e(g, g)$ is a generator of G_2

Computational Diffie-Hellman (CDH) problem. This problem states that given $g^a, g^b \in G_1$, find $g^{ab} \in G_1$. CDH assumption means that there is no probabilistic polynomial time algorithm to solve CDH problem in G_1.

A variant of CDH problem is called Reversion Computational Diffie-Hellman (RCDH) problem: given g^a, g^c, find $g^{c/a}$. In [20], the authors proved that RCDH problem is equivalent to CDH problem.

Due to the existence of bilinear map e, Decisional Diffie-Hellman (DDH) problem, i.e., given $g^a, g^b, g^c \in G_1$, decide whether $g^{ab} = g^c$ or not, can be efficiently solved in G_1 while CDH problem remains hard. G_1 is also referred as a Gap Diffie-Hellman (GDH) group.

Bilinear Diffie-Hellman (BDH) problem. It can be considered as a CDH problem in G_2. It states that, given $g^a, g^b, g^c \in G_1$, find $e(g, g)^{abc} \in G_2$. Again, BDH assumption indicates that there is no probabilistic polynomial time algorithm that can solve BDH problem in G_2.

BLS signature. Boneh et al. [21] proposed a short signature scheme based on GDH groups. An approximately 160-bit BLS signature can achieve a similar security level of a 320-bit DSA signature. The BLS signature scheme contains three algorithms, i.e., *KeyGen*, *Sign* and *Verify*, and hash function $H : \{0, 1\}^* \rightarrow G_1$ is a random oracle [22].

KeyGen. Each party chooses a random value x from $\mathbb{Z}_p^*$ (denoted by $x \xleftarrow{r} \mathbb{Z}_p^*$) as its private key; the corresponding public key is $g^x \in G_1$.

Sign. To sign a message m, the signer hashes m into G_1, i.e., $H(m)$, and computes $H(m)^x$.

Verify. Given g^x, $H(m)^x$ and m, the verifier computes $H(m)$ and checks if $e(H(m)^x, g) = e(H(m), g^x)$ holds.

Private set intersection. A private set intersection (PSI) protocol allows two parties to find the intersection of their input sets without leaking extra information (e.g., see [23], [24], [25]). A cardinality PSI protocol only allows two parties to learn the size of the intersection of their sets. In our work, we exploit a cardinality PSI protocol which is secure against the honest but curious adversary (which we will introduce later) and treat it as a black box.

B. Adversary Model

In this paper, we focus on the honest but curious adversary model and its detailed formal definitions can be found in [26]. Under this model, all users follow the specified protocol. An adversary tries to get some additional knowledge by inspecting the protocol transcripts that he gets after the protocol execution. To illustrate the security of our proposed protocols under this model, we show that a party cannot get any extra information with the protocol transcripts as well as the outputs. Note that we assume communication channels among parties are authenticated, i.e., impersonating attacks are not possible.

IV. Online Social Networks

A. Social Network Model

We model a social network as a social graph $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ where each user $u_i \in \mathcal{U}$ is represented as a single node and social relationships among them are represented in the form of edges ($\mathcal{E}$). Without ambiguity, we directly use u_i to represent u_i's identity. We first only consider friend relationships, i.e., friendships, in our model. It is not difficult to extend the proposed protocols for multi-relationships. $\mathcal{E}$ is defined as a subset of $\mathcal{U} \times \mathcal{U}$, i.e., when two users u_i and u_j establish a friendship, an edge between them is added into $\mathcal{G}$. The set $u_i.fri$ contains all u_i's friends. When u_i and u_j are friends, we have $u_j \in u_i.fri$ and $u_i \in u_j.fri$. A path from one user to another in $\mathcal{G}$ is represented by a sequence of users on this path, the number of edges on this path is defined as its depth. For example, a path from u_i to u_j with u_ℓ as the middle node is denoted by $[u_i, u_\ell, u_j]$ and it is a 2-depth path. Here, u_i is also referred as the originator of the path. Moreover, two paths are *reverse* for each other if they have same users but in reverse sequences, e.g., $[u_j, u_\ell, u_i]$ is a reverse path of $[u_i, u_\ell, u_j]$.

Besides social information, each user is equipped with some algebraic knowledge. The two previously mentioned groups $G_1 = \langle g \rangle$ and G_2 of the same prime order p together with a bilinear map $e : G_1 \times G_1 \rightarrow G_2$ and a random hash function $H : \{0,1\}^* \rightarrow G_1$ are publicly known to everyone. A key management authority assigns each user u_i a key pair (pk_i, sk_i) where the secret key is $sk_i \xleftarrow{r} \mathbb{Z}_p^*$ and its corresponding public key is $pk_i = g^{sk_i}$. When u_i and u_j become friends, u_i generates a signature $\tau_{ji} = H(fri, u_j)^{sk_i}$ for u_j as a friendship certificate. At the same time, u_j also issues u_i a friendship certificate $\tau_{ij} = H(fri, u_i)^{sk_j}$. The reason to put the identity of the user inside the certificate is to prevent users to transfer their friendship certificates to others. Each user maintains all these certificates as well as the corresponding users' identities who have issued them.

B. Access Control in Social Networks

In OSNs, a user defines an access control policy to regulate who can view or perform other actions on a certain resource that he has. We refer this user as the owner (represented by u_o) and the user who wants to access the resources as the requester (u_r). We regulate the access control policy on a certain resource is only defined by its owner, i.e., collaborative access control schemes such as [27], [28], [29], [30] are considered out of the scope of our paper. When u_r wants to access a certain resource, such as a photo or a status, he first sends u_o a request. The owner then extracts the access control policy on that resource and sends it back to u_r. Next, u_o and u_r run the protocol related to the policy. In the end, if the result of the protocol shows that u_r satisfies the policy's regulation, then u_o grants the access of the resource to u_r. As mentioned in Sect. I, we focus on two access control policies proposed in [4]. We first give their formal definitions as follows.

k-common friends. This policy regulates that the qualified requester should have at least k common friends with the owner, formally $|u_o.fri \cap u_r.fri| \geq k$.

k-depth. This policy specifies that u_o is linked with u_r through a k-depth path.

In the following two sections, we present a protocol for each of the two policies and prove its security under the honest but curious adversary.

V. k-common friends

A. Protocol Description

Our solution for k-common friends exploits the encoding scheme proposed in [24] and a cardinality PSI protocol.

In the beginning, u_r and u_o exchange random values with each other, u_r sends $R_r = g^{r_r}$ to u_o and u_o replies with $R_o = g^{r_o}$ to u_r, where $r_r, r_o \xleftarrow{r} \mathbb{Z}_p^*$. Next, both parties encode their friendship certificates using Algorithm 1.

Algorithm 1 u_i's friendship encoding scheme for u_j.

Input: r_i, R_j
Output: E_{ij} containing u_i's encodings related to u_j
1: $E_{ij} \leftarrow \emptyset$
2: **for all** $u_\ell \in u_i.fri$ **do**
3: $\quad E_{ij} \leftarrow E_{ij} \cup \{e(\tau_{i\ell}, R_j) \cdot e(H(fri, u_j), pk_\ell)^{r_i}\}$
4: **end for**
5: **return** E_{ij}

For example, suppose $u_a \in u_o.fri \cap u_r.fri$, u_o encodes the certificate $\tau_{oa} = H(fri, u_o)^{sk_a}$ into the following:

$$e(\tau_{oa}, R_r) \cdot e(H(fri, u_r), pk_a)^{r_o}. \quad \text{(I)}$$

On the other hand, u_r encodes $\tau_{ra} = H(fri, u_r)^{sk_a}$ into

$$e(\tau_{ra}, R_o) \cdot e(H(fri, u_o), pk_a)^{r_r}. \quad \text{(II)}$$

Each encoding contains two components. The first component of (I) is equal to the second component of (II) due to bilinearity of the map e, i.e.,

$$\begin{aligned} e(\tau_{oa}, R_r) &= e(H(fri, u_o)^{sk_a}, g^{r_r}) \\ &= e(H(fri, u_o), pk_a)^{r_r}. \end{aligned}$$

The same with the second component of (I) and the first one of (II). Therefore, encodings (I) and (II) are identical. Actually, the second component of one encoding is an anticipation of the first component of the other one [24]. As long as u_o and u_r have the proper friendship certificates issued by a common friend, their encodings related to this friend are identical which means they have a common element related to this common friend.

In the end, u_o and u_r perform a cardinality PSI protocol on these encodings and get the number of their common friends. Note that both u_o and u_r can choose not to encode their friendships that are considered sensitive for set intersection operations.

All the encodings are based on the friendship certificates which are already part of u_o and u_r's knowledge after they establish connections with other users. Therefore, u_o and u_r's friends do not need to participate in the process, i.e., the protocol can be executed when they are *offline*. This is an appealing feature for most situations, as it allows the protocol

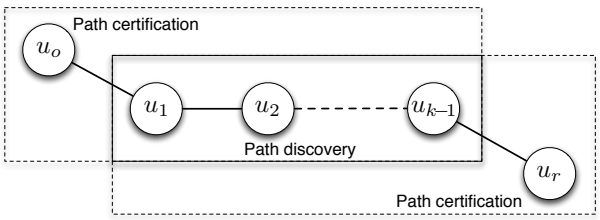

Fig. 1. The two stages of k-depth protocol.

exclusively based on u_o and u_r's local interaction and without the help of intermediate users.

B. Security Analysis

The goal of the protocol we want to achieve is that u_o (u_r) cannot learn who are friends of u_r (u_o). Note that, since u_o and u_r's friends don't participate in the protocol, they cannot cause any privacy threat.

In this protocol, except for the PSI operation, u_o and u_r only perform local computations (with their friendship certificates), i.e., they don't communicate with each other. Therefore, neither of them will get extra information from that stage. As the cardinality PSI protocol we exploit is secure against the honest but curious adversary model, our protocol is secure under this model as well.

VI. k-DEPTH

A. Protocol Description

The protocol for 2-depth policy can be implemented as a 1-common friend protocol. When the depth is bigger than 2, since neither u_o nor u_r has information about users beyond their friends, collaboration of intermediate users (neither the owner nor the requester) are needed. In the previous protocol, only u_o and u_r need to be online, common friends are discovered through friendship certificates that u_o and u_r have. We apply this idea to obtain our k-depth protocol.

Our protocol contains two stages, namely ($(k-1)$-depth) *path certification* and (k-depth) *path discovery*. As we can see from the two dashed boxes in Figure 1, in the path certification stage, u_o and u_r ask intermediate users to certify $(k-1)$-depth paths starting from them, respectively. In the path discovery stage, if a $(k-1)$-depth path originated by u_o shares a $(k-2)$-depth (reversed) path with a $(k-1)$-depth path originated from u_r (see the central solid box in Figure 1), then these two paths can compose a k-depth path between u_o and u_r. In our protocol, encodings of these two $(k-1)$-depth paths are identical. After performing a cardinality PSI protocol, a k-depth path can be discovered (if such k-depth paths exist for u_o and u_r).

Besides a key pair, each user u_i is also affiliated with a set of *depth stamps*, i.e., $s_i = \{s_i^j \mid s_i^j \xleftarrow{r} \mathbb{Z}_p^* \text{ and } j > 0\}$, and it is only known to the user himself. Here, s_i^j is the depth stamp for u_i at depth j, called u_i's j-depth stamp. We regulate that each user's 0-depth stamp is equal to 1.

1) Path certification: In this stage, u_o (u_r) first invites his friends to join the process by sending them messages. A message m is defined as a tuple (i, η, cnt, dep). Here, the randomly chosen i represents identity of the message; η is the *path certificate* which is equal to $H(u_o)$ for u_o ($H(u_r)$ for u_r) at the moment; cnt starting from 1 represents the count value; $dep = k-1$ is the length of the path. Note that everyone in the path certification stage can choose who to contact next.

Upon receiving a message with $cnt \neq dep$ from one of his friends, if an intermediate user agrees to join the path certification process, then he follows Algorithm 2.

Algorithm 2 u_j's message generation scheme

1: **receive:** $m_x = (i_x, \eta_x, cnt, dep)$ from u_i
2: **for all** $u_\ell \in u_j.fri - \{u_i\}$ **do**
3: choose a random identity i_y
4: store the link between (i_x, u_i) and i_y
5: $\eta_y \leftarrow \eta_x^{s_j^{cnt}}$
6: $cnt \leftarrow cnt + 1$
7: $m_y \leftarrow (i_y, \eta_y, cnt, dep)$
8: **send:** m_y to u_ℓ
9: **end for**

The user first remembers the links between the identity of the received message and identities of new messages that he is going to send out. Next, he generates a new path certificate by raising the old one to the power of his cnt-depth stamp. As the depth stamp is only known to the user, this operation indicates that the user agrees to certify the path. Moreover, by using his cnt-depth stamp, the user's position information on the path is directly stored into the certificate. For example, if cnt is equal to 2, then using the user's 2-depth stamp for the new path certificate shows that he is the second one on this path. This is a crucial operation in our protocol. Without it, the result in the path discovery stage may be incorrect, we will show an example later.

If a user receives a message with $cnt = dep$, then he is aware that he is the last one on the $(k-1)$-depth path. Next, he sends a *reverse message* back to the friend who sent him the message. The reverse message rm is also denoted by a tuple (i, η, σ, cnt) where i is the same as identity of the message he received; η is the $(k-1)$-depth path's certificate which will stay the same in the following processes; σ represents the *path stamp* and it is equal to $g^{s_a^1}$ at the moment if the user is u_a; cnt is reset to 2.

Algorithm 3 u_j's reverse message generation scheme

1: **receive:** $rm_y = (i_y, \eta_y, \sigma_y, cnt)$ from u_ℓ
2: find (i_x, u_i) linked with i_y
3: $\eta_x \leftarrow \eta_y$
4: $\sigma_x \leftarrow \sigma_y^{s_j^{cnt}}$
5: $cnt \leftarrow cnt + 1$
6: $rm_x \leftarrow (i_x, \eta_x, \sigma_x, cnt)$
7: **send:** rm_x to u_i

When a user gets a reverse message from his friend, he

performs the operations as specified in Algorithm 3. Since he has stored the connection between messages' identities, he is able to forward the new reverse message back to the user who sent him the corresponding message previously (step 2 in Algorithm 3). Identity information guarantees that a reverse message's forwarding path is the reverse path of the one on which the corresponding path certificate is established. Each intermediate user also builds the path stamp by raising the old one to the power of the his cnt-depth stamp. Similarly, the sequence of intermediate users are stored into the path stamp. Note that if a user uses his i-depth stamp to build a path's certificate, then he computes the path's stamp with his $(k-i)$-depth stamp. Essentially, establishment of a $(k$–$1)$-depth path's stamp simulates the building process of another $(k-1)$-depth path's certificate where these two paths together compose a k-depth path.

In the end, several $(k$–$1)$-depth path's certificates and stamps are sent back to u_o and u_r.

2) Path discovery: In this stage, u_o and u_r follow a similar procedure of 1-common friends protocol. First, they exchange two random numbers $R_o = g^{r_o}$ and $R_r = g^{r_r}$ while keeping r_o and r_r secret. Then, as the policy is k-depth, they both encode all the $(k-1)$-depth paths' certificates and stamps obtained from the last stage following Algorithm 4.

Algorithm 4 u_i's k-depth encoding scheme for u_j

Input: r_i, R_j
Output: E_{ij} containing u_i's encodings related to u_j
1: $E_{ij} \leftarrow \emptyset$
2: **for all** (η_x, σ_x) from the path certification stage **do**
3: $\quad E_{ij} \leftarrow E_{ij} \cup \{e(\eta_x, R_j) \cdot e(H(u_j), \sigma_x)^{r_i}\}$
4: **end for**
5: **return** E_{ij}

For two $(k$–$1)$-depth paths starting from u_o and u_r, respectively, if they can compose a k-depth path, then exponent of one's stamp is equal to the other one's certificate. Therefore, encodings on certificates and stamps of these two paths will be identical. In the end, a cardinality PSI protocol is performed on these encodings to find out the k-depth path.

3) An example: We present an example to show how our k-depth protocol works: the network topology is depicted in Figure 2 and the policy is 3-depth.

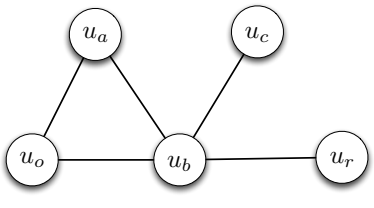

Fig. 2. An example of social network topology.

Path certification stage. In the beginning, u_o sends $m_1 = (i_1, H(u_o), 1, 2)$ to u_a and $m_2 = (i_2, H(u_o), 1, 2)$ to u_b. Upon receiving m_1, u_a chooses a random message identity i_3 and remembers the link between (i_1, u_o) and i_3. Since $cnt = 1$ in m_1, he computes a new path certificate as $H(u_o)^{s_a^1}$. Then, u_a sends $m_3 = (i_3, H(u_o)^{s_a^1}, 2, 2)$ to u_b, i.e., his only friend except u_o. Meanwhile, u_b performs similar operations and sends $m_4 = (i_4, H(u_o)^{s_b^1}, 2, 2)$ to u_a, $m_5 = (i_5, H(u_o)^{s_b^1}, 2, 2)$ to u_c and $m_6 = (i_6, H(u_o)^{s_b^1}, 2, 2)$ to u_r.

When u_b receives m_3 and finds out $cnt = dep$, he knows that he is the last one on the 2-depth path. Since cnt is equal to 2, he computes $H(u_o)^{s_a^1 s_b^2}$ and sends a reverse message $rm_3 = (i_3, H(u_o)^{s_a^1 s_b^2}, g^{s_b^1}, 2)$ back to u_a. Note that i_3 in rm_3 is identical to the identity of m_3. Similarly, u_b gets $rm_4 = (i_4, H(u_o)^{s_b^1 s_a^2}, g^{s_a^1}, 2)$, $rm_5 = (i_5, H(u_o)^{s_b^1 s_c^2}, g^{s_c^1}, 2)$ and $rm_6 = (i_6, H(u_o)^{s_b^1 s_r^2}, g^{s_r^1}, 2)$ from u_a, u_c and u_r.

Next, when u_a receives rm_3, he finds out that the identity of the message linked with i_3 is (i_1, u_o). Since cnt is equal to 2, u_a computes $g^{s_b^1 s_a^2}$ as a new path stamp and sends $(i_1, H(u_o)^{s_a^1 s_b^2}, g^{s_b^1 s_a^2}, 3)$ to u_o. On the other hand, u_b sends u_o three reverse messages that have the same identity but with different contents – $(i_2, H(u_o)^{s_b^1 s_a^2}, g^{s_a^1 s_b^2}, 3)$, $(i_2, H(u_o)^{s_b^1 s_c^2}, g^{s_c^1 s_b^2}, 3)$ and $(i_2, H(u_o)^{s_b^1 s_r^2}, g^{s_r^1 s_b^2}, 3)$ to u_o.

In the end, u_o gets four pairs of path certificate and stamp related to four different 2-depth paths starting from him, i.e.,

$$\begin{aligned}
&(H(u_o)^{s_a^1 s_b^2}, g^{s_b^1 s_a^2}) \quad \text{for path} \quad [u_o, u_a, u_b],\\
&(H(u_o)^{s_b^1 s_a^2}, g^{s_a^1 s_b^2}) \quad \text{for path} \quad [u_o, u_b, u_a],\\
&(H(u_o)^{s_b^1 s_c^2}, g^{s_c^1 s_b^2}) \quad \text{for path} \quad [u_o, u_b, u_c],\\
&(H(u_o)^{s_b^1 s_r^2}, g^{s_r^1 s_b^2}) \quad \text{for path} \quad [u_o, u_b, u_r].
\end{aligned}$$

On the other direction, u_r gets

$$\begin{aligned}
&(H(u_r)^{s_b^1 s_a^2}, g^{s_a^1 s_b^2}) \quad \text{for path} \quad [u_r, u_b, u_a],\\
&(H(u_r)^{s_b^1 s_c^2}, g^{s_c^1 s_b^2}) \quad \text{for path} \quad [u_r, u_b, u_c],\\
&(H(u_r)^{s_b^1 s_o^2}, g^{s_o^1 s_b^2}) \quad \text{for path} \quad [u_r, u_b, u_o].
\end{aligned}$$

Path discovery stage. In this stage, u_o encodes all 2-depth paths' certificate and stamp into

$$e(H(u_o)^{s_a^1 s_b^2}, R_r) \cdot e(H(u_r), g^{s_b^1 s_a^2})^{r_o}; \quad (1)$$
$$e(H(u_o)^{s_b^1 s_a^2}, R_r) \cdot e(H(u_r), g^{s_a^1 s_b^2})^{r_o}; \quad (2)$$
$$e(H(u_o)^{s_b^1 s_c^2}, R_r) \cdot e(H(u_r), g^{s_c^1 s_b^2})^{r_o}; \quad (3)$$
$$e(H(u_o)^{s_b^1 s_r^2}, R_r) \cdot e(H(u_r), g^{s_r^1 s_b^2})^{r_o}. \quad (4)$$

On the other hand, u_r encodes the information he gets into

$$e(H(u_r)^{s_b^1 s_a^2}, R_o) \cdot e(H(u_o), g^{s_a^1 s_b^2})^{r_r}; \quad (5)$$
$$e(H(u_r)^{s_b^1 s_c^2}, R_o) \cdot e(H(u_o), g^{s_c^1 s_b^2})^{r_r}; \quad (6)$$
$$e(H(u_r)^{s_b^1 s_o^2}, R_o) \cdot e(H(u_o), g^{s_o^1 s_b^2})^{r_r}. \quad (7)$$

It is clear that encoding (1) is equal to encoding (5). Paths $[u_o, u_a, u_b]$ and $[u_r, u_b, u_a]$ compose a 3-depth path between u_o and u_r (see Figure 2). After the PSI operation, u_o and u_r are aware that there exists one 3-depth path between them.

4) Discussion: We extend the main idea of k-common friends protocol to implement k-depth protocol, the two stages of our k-depth protocol do not have to be executed sequentially. Path certification stage can be a routine performed by users in the OSN once in a while, e.g., once per month. When u_r wants to access u_o's resource, both of them directly execute the path discovery stage, i.e., only u_o and u_r need to be online in our k-depth protocol. As the path certification process is a usual routine, u_o and u_r cannot agree on some

nonce, $(k-1)$-depth paths' certificates should be based on common knowledge of users. In our protocol, we use the hash value of user's identity, i.e., $H(u_o)$ and $H(u_r)$. Separation of the two stages also results in efficient communication, i.e., the first stage can be executed when the traffic in OSNs is low, and it also provides better privacy which we explain next.

B. Security Analysis

There are three parties involving in the protocol including u_o, u_r and users in the middle. For u_o and u_r, the security goal of our protocol is that we only want them to know whether there is a k–depth path between them. Extra knowledge such as who are on the path shouldn't be learned by them. For a user on the path, the security goal of our protocol is that he should only know that he is involving in a path certification stage, he shouldn't know anything more than his friend who sends him the message and the friends he will contact next.

Path certification. In this stage, what each user gets (from messages and reverse messages) are identities, count value, depth, path certificates and stamps. We analyze what information they may leak one by one. For each information, we consider the case under a single as well as multiple $(k-1)$-depth paths' certification processes. To give a clear explanation, we use k users' positions on a $(k-1)$-depth path to represent them. The user on the ith position is denoted by u_i $(0 \leq i \leq k-1)$ and u_o is at position 0.

Identity. As identities of messages are chosen randomly, they won't leak any information in both single and multiple paths' certification processes.

Count value and depth. In a $(k-1)$-depth path's certification process, a user gets his position on the path from cnt and the depth of the path from dep. Moreover, when he receives a reverse message from u_{i+1}, he knows that u_{i+1} is involved in a $(k-i)$-depth path. We argue that these information are not privacy sensitive, as a user can always guess one of his friend has another friend or a friend of friend.

However, cnt and dep may result in information leakage under several paths' certification processes originated from the same user. A user first sends a message to one of his friends with cnt and dep, later he will know how many $(dep-cnt)$-depth paths this friend originates by counting the number of reverse messages that he gets from this friend. Especially, when $cnt = dep$–1, he knows how many friends this friend has. This partial structure information can be sensitive in certain cases. To prevent this, each user should send some dummy reverse messages back which produces a noisy version of his social structure.

Path certificates and stamps. Sensitive information in certificates and stamps includes u_o's identity, intermediate users' depth keys and their connections. In a single $(k-1)$-depth path certification, what u_i gets are a partial path certificate produced by the first ith users on the path, i.e., $H(u_o)^{\prod_{j=0}^{i-1} s_j^j}$, the $(k-1)$-depth path's certificate, i.e., $H(u_o)^{\prod_{j=0}^{k-1} s_j^j}$, and a (partial) path stamp, i.e., $g^{\prod_{j=i+1}^{k-1} s_j^{k-j}}$, generated by his successors (from u_{i+1} to u_{k-1}) on the path.

With $H(u_o)^{\prod_{j=0}^{k-1} s_j^j}$, if the user is able to get $g^{\prod_{j=0}^{k-1} s_j^j}$ from another certification process, then by verifying

$$e(H(u_o)^{\prod_{j=0}^{i-1} s_j^j}, g) = e(H(u_o), g^{\prod_{j=0}^{i-1} s_j^j}),$$

he knows who is the owner (through $H(u_o)$). Similarly, $H(u_o)^{\prod_{j=0}^{i-1} s_j^j}$ may also leak u_o's identity with the relative partial path stamp. In Figure 2, u_b gets $H(o)^{s_a^1}$ from path $[u_o, u_a, u_b]$'s certification and $g^{s_a^1}$ from $[u_o, u_b, u_a]$'s certification. If u_b computes parings of these information with the above equation, he will know that the message sent from u_a is originated by u_o. To prevent this information leakage, u_o can send $H(u_o)^x$ instead of $H(u_o)$ to u_1 in the beginning where $x \xleftarrow{r} \mathbb{Z}_p^*$ is only known to himself and u_o generates different x for his different friends. In this way, even an intermediate user gets the corresponding (partial) path stamp, as he knows nothing about x, the above equation won't work. Note that before the path discovery stage, u_o needs to recover the path certificate with x^{-1}.

From $g^{\prod_{j=i+1}^{k-1} s_j^{k-j}}$ in the reverse message sent by u_{i+1}, due to the hardness of discrete logarithm problem in G_1, u_i cannot discover $\prod_{j=i+1}^{k-1} s_j^{k-j}$. Note that users who happen to be the last one on $(k-1)$-depth paths will expose their "public" 1-depth stamps when they start to forward the reverse messages and these public 1-depth stamps can be treated as their identities. For example, u_{k-1} gets $g^{s_{k-1}^1}$ which he can use to identify u_{k-1}. However, as a path stamp is computed by users with stamps of different depths, "public" 1-depth stamps will not leak their issuers' identities. For example, suppose that u_{k-3} already knows $g^{s_{k-1}^1}$ is from u_{k-1} through another path certification process. When he gets $g^{s_{k-1}^1 s_{k-2}^2}$ from u_{k-2}, as he doesn't have $g^{s_{k-2}^2}$, he cannot know that u_{k-1} is the last one on the $(k-1)$-depth path (through pairing), i.e., u_{k-1} and u_{k-2} are friends. Moreover, suppose that u_{k-2} even knows that $g^{s_{k-1}^1 s_{k-2}^2}$ is built by u_{k-2} and u_{k-1}, he cannot get u_{k-2}'s 2-depth stamp s_{k-2}^2 from $g^{s_{k-1}^1 s_{k-2}^2}$ and $g^{s_{k-1}^1}$ due to the RCDH assumption in G_1.

Now, suppose that u_i and u_{i+x} $(i+x \leq k-1)$ are friends, i.e., a circle appears in the path. When u_i joins the process and sends a message to u_{i+1}, u_{i+1} then contacts u_{i+2}, so on and so forth. Later, u_{i+x} sends a message to u_i. Since u_i has no information about his successors' depth keys, he doesn't know that the message he receives from u_{i+x} is based on the message he sends to u_{i+1} before. Therefore, u_i doesn't know that u_{i+1} and u_{i+x} are linked through a $(x-1)$-depth path.

However, with several path's certification processes, sensitive information can be disclosed through certificates and stamps. Suppose u_o's two friends are linked by a $(k-2)$-depth path, i.e., u_o is in a k-depth circle. Later, when he gets path certificates and stamps on two $(k-1)$-depth paths which can compose the circle from these two friends, as the circle is also a k-depth path, by performing bilinear map, he can get whether these two friends are connected by a $(k-2)$-depth path. In the example above, u_o is linked with $[u_o, u_a, u_b]$ and $[u_o, u_b, u_a]$, by paring path certificates and stamps on these two paths, he has $e(H(u_o)^{s_a^1 s_b^2}, g^{s_b^1 s_a^2}) = e(H(u_o)^{s_b^1 s_a^2}, g^{s_a^1 s_b^2})$ which indicates that u_a and u_b are friends. We propose a

simple solution for this leakage. Now, the protocol regulates that when a user receives a message with $dep=2$ and $cnt=1$, he only sends new messages to his friends who haven't sent him a message with $dep=2$ and $cnt=2$ yet. In Figure 2, after u_a and u_b receive messages from u_o, suppose that u_a first sends $m_3=(i_3, H(o)^{s_a^1}, 2, 2)$ to u_b. Later, when u_b wants to send new messages to his friends, as he finds out that u_a already sent him m_3 with $dep=2$ and $cnt=2$, he only sends new messages to u_c and u_r. In the end, u_o won't know that u_a and u_b are connected. However, since there is no information in a message about the originator of the path, our solution reduces chances for finding paths. For example, the message sent from u_a to u_b may come from another user than u_o. Also, our solution only supports 2-depth path certification for 3-depth policy. Although the information that a user knows his two friends are linked with a 5-depth path is not that valuable, protecting two users' private links under 3 depths is still necessary. We leave the general protection scheme as a future work.

Path discovery. In this stage, only u_o and u_r participate the protocol. First, as the two stages are independent, no intermediate users knows who is the owner or requester. Moreover, intermediate users do not know if there will be a run of the path discovery stage. Second, as the cardinality PSI protocol is secure against honest but curious adversaries, u_o and u_r only get whether a qualified path exists or not, nothing more. Especially, u_o also doesn't know which friend of his is on the k-depth path. The same holds for u_r.

C. Multi-relationship k-depth Protocol

Our k-depth protocol can be extended to support multi-relationships. We first introduce the multi-relationship social network model. Let the set $\mathcal{R}$ contain all the relationship types. Only symmetric relationships, such as friend and colleague, are considered. The social network is defined as a graph $\mathcal{G}=(\mathcal{U},\mathcal{E})$, where $\mathcal{E}$ now is denoted as a subset of $\mathcal{U}\times\mathcal{R}\times\mathcal{U}$, i.e., each edge is labeled with a relationship type. A k-depth access control policy regulates that u_o is linked with u_r through a k-depth path where each edge has a certain relationship type. All k relationship types (from u_o to u_r) can be represented as a k-tuple $(rp_1, rp_2, \ldots, rp_k)$ where $rp_i \in \mathcal{R}$ $(1 \le i \le k)$. Note that these k relationships do not have to be distinct from each other.

Our multi-relationship k-depth protocol also contains two stages. The path discovery stage remain the same while there are two differences related to the path certification stages. First, a *relationship chain* is added in each message and its function is to inform intermediate users which social links to contact next. A relationship chain generated by u_o is defined as $\langle rp_1, \ldots, rp_{k\text{-}1}\rangle$ which contains the first $(k$-1)-th relationship types specified in the policy with the same sequence. Moreover, the first relationship in a chain is defined as the *tail* of the chain. On the other direction, the relationship chain generated by u_r is in a reverse order, i.e., $\langle rp_k, \ldots, rp_2\rangle$. When a user receives a message, he will delete the tail from the chain and send new messages to his social links who are in the new tail of the chain with him. The second difference is that we have to integrate the relationship type into paths' certificates and stamps. Instead of one set, each user should have different sets of depth stamps for different relationship types. When a user receives a message, he uses his cnt-depth stamp from the stamp set related to the tail of the chain to compute the new certificate. The same procedure applies for computing the path stamp. Note that the relationship type a user integrates into a path's certificate (stamp) is always the one that he is in with the user who sent him the message (reverse message). This guarantees that encodings related to two $(k-1)$-depth paths that can compose a k-depth path in discovery stage are identical.

TABLE I
COMPARISON OF k-DEPTH PROTOCOLS.

	[17]	[16]	Our work
Intermediate user offline	✓		✓
Multi-relationships			✓
Computation cost	Hash	Hash	Paring
Communication steps	k	k	$k-1$
Honest but curious model	Partially	Partially	✓

D. Comparison with Existing Schemes

We compare our k-depth protocol with the schemes proposed in [17], [16] (see Table I). The solution in [17] and our protocol contain two independent stages, only u_o and u_r need to be online when finding the path. On the other hand, the protocol proposed in [16] requires all the intermediate users to be online. Different from ours, the two protocols [17], [16] do not support multi-relationships.

Messages passing among users in their schemes are based on hash functions, this is more efficient than the bilinear map. On the other hand, our protocol consumes one step of communications less when finding the paths (each user certifies $(k$–$1)$-depth paths, instead of k-depth) than theirs which save a large number of operations. More precisely, suppose that each user has in average n friends, to find a $(k-1)$-depth path, totally $2(k-1)n^{k-1}$ times user-to-user communications are consumed (see Sect. VII), while the number is kn^k in both works. Moreover, as each communication step needs computations, a large number of computations (mainly exponentiations) are saved in our protocol as well.

Since their tokens are built through a publicly known hash function, sensitive information can be leaked. For example, as mentioned in [17], a user can know whether a token he receives is based on another token sent by him previously. This indicates his corresponding two friends are linked. The same threat happens to the scheme in [16]. However, this information leakage can be prevented in our protocol as we explained in the security analysis.

VII. PERFORMANCE ANALYSIS

In this section, we first give a formal efficiency analysis of our protocols, then present empirical results on the protocols through a Facebook dataset [31].

TABLE II
THEORETICAL PERFORMANCE ANALYSIS.

Computation	**k-common fri.**	**k-depth**
Paring	$4n$	$4n^{k-1}$
Hash	G_1:$2n$	G_1:$2n^{k-1}$
Multiplication	G_2:$2n$	G_2:$2n^{k-1}$
Exponentiation	G_2:$2n$	G_1:$2(k-1)n^{k-1}$ G_2:$2n^{k-1}$
PSI	$O(n \log\log n)$	$O(n^{k-1} \log\log n^{k-1})$
Communication	**k-common fri.**	**k-depth**
User-to-user	2	$2(k-1)n^{k-1}$
PSI	$O(n)$	$O(n^{k-1})$

A. Theoretical Efficiency Analysis

For each of our protocols, we analyze its computation and communication complexities (see Table II). We assume that each user's average number of friends is n. As friendship establishments are normal routines, we do not consider them as part of our protocols. Moreover, for k-common friends and k-depth protocols, we exploit the PSI scheme proposed in [32] to give a general complexity for our protocols.

1) Computation Cost: **k-common friends.** Computations are performed in both encoding stage and PSI protocol. To encode a friendship certificate, u_o (u_r) needs to perform two pairing computations (each encoding contains two components), one hash function operation in G_1, one exponentiation and one multiplication in G_2. Since there are totally $2n$ friends for u_o and u_r, $4n$ times of parings, $2n$ hashes in G_1, $2n$ exponentiations and $2n$ multiplications in G_2 are needed. Inputs for PSI operations are $2n$ encodings, computation related to set intersection can be finished in $O(n \log\log n)$ time [32].

k-depth. Since the two stages of our protocol are independent, for the path certification stage, we only consider computation and communication consumption of a single user. In this stage, computations are mainly exponentiations in G_1 for path certificates and stamps. For a $(k-1)$-depth path staring from u_o, as $k-1$ users compute the path certificate and stamp by exponentiation, totally $2(k-1)$ exponentiations are needed, i.e., $k-1$ for path certificate and $k-1$ for path stamp. For the whole stage, u_o can get maximal n^{k-1} pairs of path certificate and stamp. Therefore, the computation cost for a single user is $2(k-1)n^{k-1}$ exponentiations.

For the path discovery stage, there are maximal $2n^{k-1}$ path certificates need to be encoded for both u_o and u_r, still each encoding needs two parings, one hash in G_1, one exponentiation and one multiplication in G_2. Therefore, totally $4n^{k-1}$ paring operations, $2n^{k-1}$ hashes in G_1, $2n^{k-1}$ exponentiations and $2n^{k-1}$ multiplications in G_2 are needed. Again, with $2n^{k-1}$ path certificates as inputs, by adopting the PSI protocol in [32], computation complexity for finding common encodings is $O(n^{k-1} \log\log n^{k-1})$.

2) Communication Cost: **k-common friends.** Communications are needed in two operations. The first one is exchanging two random values in the beginning, where two user-to-user communications are needed. The second communication consuming operation is related to the PSI protocol. As the PSI protocol has constant rounds and its inputs are $2n$ encodings, it needs $O(n)$ user-to-user communications.

k-depth. As mentioned before, a user can get maximal n^{k-1} pairs of path certificate and stamp in path certification stage. Each pair requires $2(k-1)$ user-to-user communications. Totally $2(k-1)n^{k-1}$ communications are needed. Path discovery stage is similar to k-common friends protocol, its communication complexity is $O(n^{k-1})$.

TABLE III
DATASET SUMMARY.

Number of nodes	63,731
Number of edges	1,634,180
Average degree	25.6
Average clustering coefficient	0.253
Number of connected components	144
Number of triangles	3,501,542

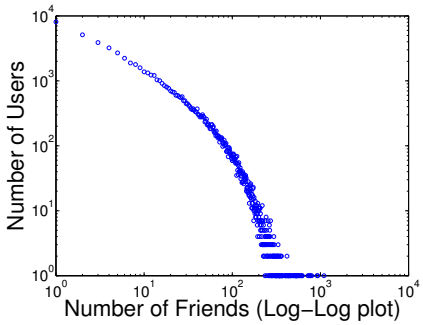

Fig. 3. Node distribution of the dataset.

B. Empirical Efficiency Analysis

Dataset. We use the Facebook dataset collected by the authors of [31] to perform our experiments, the dataset is summarized in Table III. In Figure 3, we plot the number of users as a function of users' number of friends, as we can see, users' friends number, i.e., degree in social graph, follows a power-law distribution. Most of users have a small number of friends (around half of users have less than 10 friends) while only a few users have a large number of connections. This indicates that most of users won't need to perform a huge amount of computations when running our protocols.

Experiment setup. Our experiments were conducted on a 64-bit Linux system with an Intel Core i7 1.80GHz×4 and 8GB RAM. We implement our protocols using the MIRACL Cryptographic SDK [33]. We choose Barreto-Naehrig Curve (security level AES-128) as the pairing curve. Since there are many existing PSI protocols and implementations, we can adopt any of them. For example, the scheme in [25] can perform set intersection operations on two million-element sets within 41 seconds. Therefore, we only focus on the performance of our protocol before the execution of PSI, where pairing operations dominate the computation cost. In our experiments, performing a pairing takes around 6 ms.

We randomly sampled 2,000 users from the dataset for k-common friends, 3-depth protocol and 4-depth, respectively. As the average path length between any two users (from more

TABLE IV
SAMPLE USERS' FEATURE SUMMARY.

	k-common	3-depth	4-depth	4-depth (100)
Total	49,665	4,398,980	511,424,020	27,163,893
Average	24.8	2,199.5	255,710.0	271,640.9
S.D.	40.7	5,067.2	613,000.1	733,250.8
Max.	570	92,748	7,213,810	4,514,853

than 1 billion users of Facebook) is 4.7 [34], 5-depth protocol is neither necessary nor likely to be performed. Therefore, we only evaluated k-depth protocol with $k = 3, 4$. Moreover, in reality the 3-depth protocol is exploited more often than the 4-depth protocol. The number of friends as well as the number of 2-depth and 3-depth paths for the sampled users are summarized in Table IV. Note that we only chose 100 users to perform 4-depth protocol for the purpose of illustration. These 100 users features are summarized in the last column of Table IV: its mean and standard deviation value are not that different from the sample of 2,000 users for 4-depth protocol. We notice that all the sample data have large std. value, this is due to the power law distribution of users' friends number, see Figure 3.

Evaluation results. The experiment results are presented in Table V. For k-common friends, the average time for encoding all friendship certificates is 0.135 second while the worst case (the user who has 570 friends, see Table IV) takes only 3 seconds. Due to the power law distribution of friends number, almost half of users (48.1%) can finish their protocol in less than 0.05 second, more than 94% users can finish in 0.5 second. For 3-depth protocols, the average running time is around 16 seconds. More than half of the 3-depth protocols (1,121/2,000) can finish within 5 seconds. The average running time of the 4-depth protocols is about 33.6 minutes. In fact, 34 users (out of 100) can finish their protocol in 60 seconds; nearly half of them (47/100) can finish in less than 3 minutes; and about 70% of the users can finish in less than 10 minutes. As described in Section VI, path certification can be treated as a normal routine, thus it doesn't have to be counted as part of k-depth protocol. Moreover, a user can choose not to join a k-depth protocol, or not to send messages to all his friends, meaning that the computations needed in practice can be further reduced. Another way to improve the performance of the k-depth protocols is to use more efficient paring implementation, such as the inline assembly code of MIRACL.

Discussion on k-depth protocols. As discussed in Section VI, although we use expensive pairing operations, our k-depth protocol's performance is still comparable with the protocol proposed in [17] which mainly uses hash functions. First, our k-depth protocol uses one less step for communications, thus a big amount of computation overhead can be saved. As presented in Table IV, the total number of 3-depth paths (for the 4-depth protocol) for the sample users is 100 times larger than the number for 2-depth paths (for the 3-depth protocol). Therefore, the scheme in [17] needs to perform at least 100 times more operations, i.e., hash functions as well as communications than ours. Second, in the path certification

TABLE V
TIME CONSUMPTION SUMMARY (SEC).

k-common friends					
Average	0.135	S.D.	0.222	Worst case	3.078
Time	≤ 0.01	$(0.01, 0.05]$	$(0.05, 0.1]$	$(0.1, 0.5]$	> 0.5
% users	13.65	34.45	14.90	31.65	5.35
3-depth					
Average	16.263	S.D.	37.447	Worst case	684.401
Time	≤ 1	$(1, 5]$	$(5, 10]$	$(10, 20]$	> 20
% users	30.95	25.10	11.95	11.50	20.50
4-depth					
Average	2,015.185	S.D.	5,372.101	Worst case	32,833.750
Time	≤ 60	$(60, 180]$	$(180, 600]$	$(600, 1200]$	> 1200
% users	34.00	13.00	23.00	7.00	23.00

stage, intermediate users can choose not to join in our protocol. Hence, the user will get a subset of all the 3-depth paths in the end.

In details, users in the first protocol of [17] need to build an "imaginary" hash tree, and the number of descendants of each node is the number of maximal degree of a user in the social network. This will be a huge tree with a lot of redundant nodes. In the dataset that we use, the maximal node degree is 1,098, meaning that the tree a user needs to build for a 3-depth protocol will have $|u.fri| \times 1098 \times 1098$ nodes, while in our experiments each user only needs to perform around 2,200 times pairing products on average. For example, for a user with only 25 friends, to perform $25 \times 1098 \times 1098$ times SHA-256 function in MIRACL, it needs around 29 seconds while in our scheme the computation only needs 16 seconds. In the extended scheme of [17], the user can build a more accurate hash tree. However, this extended scheme is only designed for discovering 3-depth paths.

VIII. CONCLUSION

In this paper, we addressed the challenge on how to enforce relationship-based access control policies on decentralized social networks. To this end, we have provided privacy-preserving protocols for two types of access control policies, i.e., k-common friends and k-depth. While the protocol for k-common friends is new, our k-depth protocol has better communication complexity and security than the existing solutions. Through experiments on a Facebook dataset, we illustrate that our protocols are efficient in practice.

There are a few research directions for the future. Firstly, it is interesting for us to re-evaluate the performance of our protocols with some more recent social network dataset than the current dataset [31] used in this paper. Secondly, we want to make our k-depth protocol secure under the malicious adversary model [26] and design protocols for other policies such as 'clique' and 'celebrity' [4]. It is also important for us to investigate whether it is possible to extend our protocols to cope with friendship revocation by incorporating some ideas, for example, from [35], [36].

ACKNOWLEDGMENT

We would like to thank Qiang Tang for his insightful comments.

REFERENCES

[1] P. W. Holland and S. Leinhardt, "Transitivity in structural models of small groups." *Comparative Group Studies*, 1971.

[2] C. E. Gates, "Access control requirements for Web 2.0 security and privacy," in *Proc. IEEE Workshop on Web2.0 Security and Privacy (W2SP)*, 2007.

[3] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham, "A semantic web based framework for social network access control," in *Proc. 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 2009, pp. 177–186.

[4] P. W. L. Fong, M. M. Anwar, and Z. Zhao, "A privacy preservation model for Facebook-style social network systems," in *Proc. 14th European Symposium on Research in Computer Security (ESORICS)*, ser. LNCS, vol. 5789. Springer, 2009, pp. 303–320.

[5] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam, "PrPl: A decentralized social networking infrastructure," in *Proc. 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010, pp. 1–8.

[6] L. A. Cutillo, R. Molva, and M. Önen, "Safebook: A distributed privacy preserving online social network," in *Proc. 12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. IEEE CS, 2011, pp. 1–3.

[7] C.-m. A. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee, "Decentralization: The future of online social networking," in *W3C Workshop on the Future of Social Networking Position Papers*, vol. 2, 2009, pp. 2–7.

[8] B. Carminati, E. Ferrari, and A. Perego, "Enforcing access control in web-based social networks," *ACM Transactions on Information & System Security*, vol. 13, no. 1, p. Article No. 6, 2009.

[9] P. W. L. Fong and I. Siahaan, "Relationship-based access control policies and their policy languages," in *Proc. 16th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 2011, pp. 51–60.

[10] P. W. L. Fong, "Relationship-based access control: protection model and policy language," in *Proc. 1st ACM Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 2011, pp. 191–202.

[11] G. Bruns, P. W. L. Fong, I. Siahaan, and M. Huth, "Relationship-based access control: its expression and enforcement through hybrid logic," in *Proc. 2nd ACM Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 2012, pp. 117–124.

[12] J. Pang and Y. Zhang, "A new access control scheme for Facebook-style social networks," in *Proc. 9th Conference on Availability, Reliability and Security (ARES)*. IEEE CS, 2014, pp. 1–10.

[13] Y. Cheng, J. Park, and R. S. Sandhu, "Relationship-based access control for online social networks: beyond user-to-user relationships," in *Proc. 4th IEEE Conference on Information Privacy, Security, Risk and Trust (PASSAT)*. IEEE CS, 2012, pp. 646–655.

[14] B. Carminati and E. Ferrari, "Privacy-aware collaborative access control in web-based social networks," in *Proc. 22nd IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC)*, ser. LNCS, vol. 5094. Springer, 2008, pp. 81–96.

[15] ——, "Enforcing relationships privacy through collaborative access control in web-based social networks," in *Proc. 5th Conference on Collaborative Computing (CollaborateCom)*. IEEE CS, 2009, pp. 1–8.

[16] M. Xue, B. Carminati, and E. Ferrari, "P3D - privacy-preserving path discovery in decentralized online social networks," in *Proc. 35th IEEE Computer Software and Applications Conference (COMPSAC)*. IEEE CS, 2011, pp. 48–57.

[17] G. Mezzour, A. Perrig, V. Gligor, and P. Papadimitratos, "Privacy-preserving relationship path discovery in social networks," in *Proc. 8th Conference on Cryptology and Network Security (CANS)*, ser. LNCS, vol. 5888. Springer, 2009, pp. 189–208.

[18] M. Backes, M. Maffei, and K. Pecina, "A security API for distributed social networks," in *Proc. 18th Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society, 2011, pp. 35–51.

[19] K. B. Frikken and P. Srinivas, "Key allocation schemes for private social networks," in *Proc. 8th ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2009, pp. 11–20.

[20] X. Chen, F. Zhang, and K. Kim, "A new ID-based group signature scheme from bilinear pairings," in *IACR ePrint Archive: Report 2003/116*, 2003.

[21] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Proc. 7th Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, ser. LNCS, vol. 2248. Springer, 2001, pp. 514–532.

[22] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Proc. 1st ACM Conference on Computer and Communications Security (CCS)*. ACM, 1993, pp. 62–73.

[23] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Proc. 23rd Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, ser. LNCS, vol. 3027. Springer, 2004, pp. 1–19.

[24] E. Stefanov, E. Shi, and D. Song, "Policy-enhanced private set intersection: sharing information while enforcing privacy policies," in *Proc. 15th Conference on Practice and Theory in Public Key Cryptography (PKC)*, ser. LNCS, vol. 7293. Springer, 2012, pp. 413–430.

[25] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in *Proc. 20th ACM Conference on Computer and Communications Security (CCS)*. ACM, 2013, pp. 789–800.

[26] O. Goldreich, *The Foundations of Cryptography - Volume 2*. Cambridge University Press, 2004.

[27] A. C. Squicciarini, M. Shehab, and F. Paci, "Collective privacy management in social networks," in *Proc. 18th Conference on World Wide Web (WWW)*. ACM, 2009, pp. 521–530.

[28] Y. Sun, C. Zhang, J. Pang, B. Alcalde, and S. Mauw, "A trust-augmented voting scheme for collaborative privacy management," in *Proc. 6th Workshop on Security and Trust Management (STM)*, ser. LNCS, vol. 6710. Springer, 2011, pp. 132–146.

[29] ——, "A trust-augmented voting scheme for collaborative privacy management," *Journal of Computer Security*, vol. 20, no. 4, pp. 437–459, 2012.

[30] H. Hu, G.-J. Ahn, and J. Jorgensen, "Multiparty access control for online social networks: model and mechanisms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 6, pp. 341–354, 2013.

[31] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proc. 2nd ACM SIGCOMM Workshop on Social Networks (WOSN)*. ACM, 2009, pp. 37–42.

[32] C. Hazay and K. Nissim, "Efficient set operations in the presence of malicious adversaries," in *Proc. 13th Conference on Practice and Theory in Public Key Cryptography (PKC)*, ser. LNCS, vol. 6056. Springer, 2010, pp. 312–331.

[33] "Miracl crypto library," http://www.certivox.com/miracl/.

[34] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The anatomy of the facebook social graph," *CoRR*, vol. abs/1111.4503, 2011.

[35] S. Jahid, P. Mittal, and N. Borisov, "EASiER: Encryption-based access control in social networks with efficient revocation," in *Proc. 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. ACM, 2011, pp. 411–415.

[36] S. Preibusch and A. R. Beresford, "Establishing distributed hidden friendship relations," in *Proc. 17th Workshop on Security Protocols (SPW)*, ser. LNCS, vol. 7028. Springer, 2013, pp. 321–334.