A Collaborative Framework for Privacy Protection in Online Social Networks

Yan Zhu[1,2], Zexing Hu[1], Huaixi Wang[3], Hongxin Hu[4], Gail-Joon Ahn[4]
[1]Institute of Computer Science and Technology, Peking University, Beijing 100871, China
[2]Key Laboratory of Network and Software Security Assurance (Peking University), Ministry of Education, China
[3]School of Mathematical Sciences, Peking University, Beijing 100871, China,
[4]Laboratory of Security Engineering for Future Computing (SEFCOM), Arizona State University, Tempe, AZ 85287, USA
Email: {yan.zhu,huzx,wanghx}@pku.edu.cn, {hxhu,gahn}@asu.edu

September 17, 2010

Abstract

With the wide use of online social networks (OSNs), the problem of data privacy has attracted much attention. Several approaches have been proposed to address this issue. One of privacy management approaches for OSN leverages a key management technique to enable a user to simply post encrypted contents so that only users who can satisfy the associate security policy can derive the key to access the data. However, the key management policies of existing schemes may grant access to unaurhorized users and cannot efficiently determine authorized users. In this paper, we propose a collaborative framework which enforces access control for OSN through an innovative key management focused on *communities*. This framework introduces a community key management based on a new group-oriented convergence cryptosystem, as well as provides an efficient privacy preservation needed in a private OSN. To prove the feasibility of our approach, we also discuss a proof-of-concept implementation of our framework. Experimental results show that our construction can achieve the identified design goals for OSNs with the acceptable performance.

1 Introduction

Online social networks (OSNs) have become an important web service where people can publish and share resources (personal tastes, blogs, or viewpoints) through different types of relationships [1]. A number of social network sites have recently emerged and they are becoming a popular and useful approach in people's daily life. For example, people can make friends with Facebook (http://www.facebook.com) or MySpace (http://www.myspace.com), find job information in LinkedIn (http://www.linkedin.com), and so on. The availability of information brings convenience to modern life while significantly raising issues related to personal privacy. For instance, personal private data may be used for promoting unnecessary products, and resources may be abused by some unauthorized users, etc.

It is crucial to effectively protect user privacy in OSN. A significant amount of work for privacy protection on OSN has been introduced [2–4]. For example, flyByNight [2] is a Facebook application designed to protect the privacy of messages exchanged between Facebook users. NOYB (short for "None Of Your Business") [5] is another system targeted at cryptographically protecting user privacy on Facebook. Persona [3] is a private OSN which encrypts user data with attribute-based encryption (ABE), allowing users to apply fine-grained policies over users who may view their data. Although some new techniques were introduced in these solutions, it is still necessary for a centralized server to enforce access control, which cannot protect the privacy of users against the centralized server. Also, some solutions implemented access control at client-side but their approach should be synchronous, requiring multiple users to be online simultaneously.

One of efficient ways for enforcing access control in OSN is to allow users to put the encrypted data on the server and then only the users who can derive the decryption key would decrypt and access the data. Normally, it can be performed through key management. The advantage of this approach is that a user just simply posts her content but the unauthorized users are not able to obtain the key. Some schemes based on

Table 1: Comparison between two existing OSN shcemes and our scheme.

	flyByNight [2]	Persona [3]	Our scheme
Cryptosystem	ElGamal/Proxy encryption	PKC/ABE	ElGamal/GCC
Autonomy	ElGamal managed by system manager; Proxy Encryption by application proxy	PKC managed by system manager; ABE managed by group creator	Full autonomy
Independence	Yes	Yes	Yes, a set of trusted users
Collaboration	No	No	Yes
Anonymous authentication	No	No	Yes
Revocation	No	No	Yes
Integrity checking	No	No	Yes
Relationship transitive	By group manager	From friend to friend	From friend to friend
Post message encryption	One-time by client-side;Each download by application proxy	One-time by client-side	One-time by client-side

this idea have been proposed [6–8]. However, these schemes based on traditional cryptographic techniques have limitations when dealing with multiple groups in OSN since either users must store multiple copies of encrypted data but are unable to give data based on membership in multiple groups, or users must know the identities of everyone to whom they give access.

We believe that a practical and effective key management access control scheme should provide the following properties: 1) *Autonomy*, once a user joins in a private OSN, he chooses his public key and private key by himself and the OSN manager cannot obtain his private key; 2) *Independence*, a community is constructed by a set of trusted users and there is no third party involved; 3) *Collaboration*, the kernel members can collaborate to construct and maintain a private OSN so as to reduce the maintenance complexity; 4) *Anonymous Authentication*, OSN can verify the validity of the user's access permission for a private OSN without a user's identity; and 5) *Revocation*, a community could revoke the permission of authorized users permanently or temporarily.

1.1 Our Contributions

To meet the privacy needs of OSN, we present a solution. which fulfills above-mentioned requirements. Our collaborative framework can provide flexible, efficient privacy protections needed in a private OSN without the intervention of a system manager. We briefly summarize the contributions of our work in this paper.

- We propose a system architecture for a private OSN. In this architecture community creators can collaborate to manage and maintain their communities. There is no need for a centralized management server to build PKC/PKI for key exchange and to monitor the behavior of all users;
- We provide a community key management method for our architecture based on a new group-oriented convergence cryptosystem (GCC). This method leverages the following properties: the community is built on convergence of some users' private keys, the upload and download of resources provide the authentication and integrity checking, as well as there exist efficient mechanisms for access permission delegation and sophisticated revocation; and
- To prove the feasibility of our architecture, a proof-of-concept prototype of the proposed approach is implemented by constructing a GCC cryptosystem and an application of community key management method. Experimental results show that our construction can achieve the identified design goals for protecting privacy in OSN with the acceptable performance.

Table 1 summarizes the comparison results between flyByNight [2], Persona [3], and our scheme. We can observe that our approach have following advantages: autonomy, collaboration, anonymous authentication, revocation, and integrity checking. These features could significantly mitigate privacy risks in using OSNs.

1.2 Organization

The rest of this paper is organized as follows. We describe the common cryptographic techniques for OSN and how to comprise better cryptosystems in Section 2. We discuss the system architecture of our private OSN in Section 3. We introduce the preliminaries of our GCC scheme and present our basic construction for community key management in Section 4. Section 5 discusses how the proposed approach can be realized in a practical application. We describe the related work in Section 6 followed by the conclusion in Section 7.

2 Cryptography in OSNs

The main task of cryptography in building a private OSN is to restrict the information available in an appropriate range. We make use of relationships or social links to represent this range in a social network. For example, family, neighbor, co-worker, boss, teammate, and other relations might define such a relationship in a private OSN. In this paper, the relationship is simply termed "friend".

Encrypting sensitive information to protect it from misuse is hardly a new concept, but the OSN setting is different from typical group scenarios. One of the differences is that the sender may not be in charge of group membership. For example, Alice may post a message on Bob's wall, encrypted for Bob's friends, without (necessarily) knowing the list of Bobs friends. Another aspect of the OSN setting is that the number of potential users might be very large in a group. These two features lead to the deployment and management of data in OSN arduous.

2.1 Limitations of Common Encryption Approach

In order to construct A private OSN setting, serval schemes have been proposed in recent years . Although these schemes adopted different cryptographic techniques, such as traditional symmetric/asymmetric encryption [6, 7, 9, 10], as well as attribute-based encryption (ABE) [3, 11, 12], they have a same working model: To create a new group from a list of known friends, Alice (the creator) encrypts a newly-generated group key with the public key of each member of the new group (obtained from PKC/PKI). She then distributes this key to the members of that group and uses the key to encrypt messages for the group. The information sharing can be realized by exchanging the key within the groups. In this model, the group key may be symmetric, in which case only group members can encrypt for the group, or asymmetric, which allows non-members to encrypt as well.

Although it looks as though this model has a simple structure, it in fact requires not only the establishment of a PKC/PKI system, but also requires a tedious task of key management for the creator. Moreover, the users' public keys based on PKC/PKI are used to distribute the group key in this model. This means that all members in a social network are managed and monitored by a centralized management server. In addition, this model has several usability and security issues:

- The user needs to store many community keys if he/she belongs to several communities;
- There does not exist an efficient way to revoke the member permanently or temporarily; and
- There does not exist an efficient way for anonymous authentication with the view of tracing the behavior of users and computer forensics.

Moreover, a group creators must carefully carry out various maintenance work, such as deleting obsolete information, keeping undesired readers off, and putting hot topics in order, and so on.

2.2 Our Approach

Our approach, to protect sensitive information in web services from unauthorized access, is to encrypt information using user-controlled keys and to provide access to data using user-controlled delegation. This approach is constructed on a new group-oriented convergence cryptosystem (GCC), which implements encryption and authentication for groups. The most striking feature of this cryptosystem is that this system is organized

and managed in a spontaneous way without a system manager. That is, a group of trusted users, not one user, collaborate to manage and maintain a private community. Moreover, this cryptosystem does not need a PKC/PKI system to realize the exchange of group key.

To use GCC, each user in OSN generates the user's private key by himself and registers a public label into the OSN. To create a community, some known users with the same interest (called as the creators of community) generate a community key (CK) in a cooperation way. All of the creators' private keys are valid for this community key. For each friend, a user can then generate an access permission key (APK) corresponding to his own private key and the friend's public label. Using the private key and the community's APK, the user can decrypt (or access) the shared information, but not encrypt (or publish) the information into the community. The encryption operation cannot be implemented unless a user holds the community key.

In order to avoid the adoption of PKC/PKI systems, a temporary public key generated from a user's private key can be used to realize the exchange of encrypted key. In addition, there exists an efficient authentication protocol, by which a untrusted storage service provider (SSP) can check whether or not a user belongs to a certain community.

Furthermore, in our model each user in OSN has only one private key. Each time the user joins in a community, she will be assigned an APK key from her friends, but this APK is invalid for other users. This approach can effectively prevent security problems caused by the loss of access permission key.

3 Our System Architecture

In this section, we introduce a private OSN architecture based on a group-oriented convergence cryptosystem. In this architecture, a predominant method of sharing data in OSNs is via collaborative applications.

3.1 Community and Member Category

Before we describe the framework of our scheme, we first introduce an important concept – community, which is the core notion of our approach. Following the traditional definition of social network, a community is a loose collection of users with the same interest. In our private OSN, a community is organized and managed by collaborative Web applications. By joining a community, one gains the right to create new contents in this community and access others' contents. For instance, the quintessential Facebook application, the Wall, is a peruser forum that features posts and comments from the user and his friends; the Facebook Photos application stores comments and tags for each picture and displays them to friends; and the Flickr photo management and sharing application allows each photograph has a page where members of the Flickr community can comment on photographs.

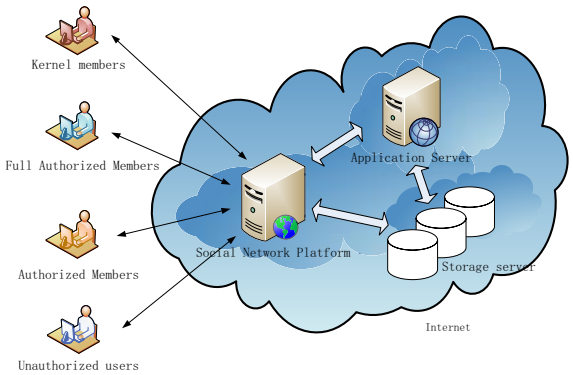


Figure 1: The system architecture for a private OSN.

We introduce a generic model to implement above-mentioned collaborative OSN applications. Figure 1 shows a system architecture for our model. In this architecture, a third party is required to be responsible for the web-based applications, as well as the storage of published data. Meanwhile, it also provides some

services for users, such as Web browser service. But we do not demand that this third party is credible for a private OSN. Existing social network sites, such as Facebook, Flickr, and Myspace, even cloud computing platforms are appropriate environments applying our model.

In order to define the range of access control in a private OSN, we classify the users in social networks into four categories:

- Kernel members (KM): can create and manage a special community by collaboration and have rights to publish, delete, access or update resources released by other members of the community;
- Full authorized members (FAM): have full rights to publish and access resources in the community, but do not have permissions to delete or update resources;
- Authorized members (AM): can access the resources by using her own access permission, but cannot publish these resources;
- Unauthorized users (UU): may not have permissions to access resources published by community members.

Note that, it is technically possible using the "delete" and "update" operations to compromise the security of a community, for example, the malicious member can make use of them to manipulate or forge others' opinions. Hence, it is necessary for authorized members to restrict their maintenance operations only for kernel members. Moreover, it is critical to adopt an efficient authentication method to distinguish kernel members from the others.

3.2 Our Model and Architecture

Our private OSN model could be built in existing social network platforms, such as Facebook, Orkut, etc, which usually allow developers to create "applications" to extend the types of information that can be stored, manipulated, and shared using social network interfaces. Fig. 2 depicts a application dataFlow for our architecture. In this model, an OSN Platform API acts as a middleware for all interactions between application providers and end users. End users, including kernel members, (full) authorized members, and unauthorized users, initiate contact with an application provider through a URL on OSN platforms. The platforms interpret input data along with these requests and pass their interpreted data via the Internet to the application servers, whose addresses are registered with platforms by the application developers. The application server then performs requested actions based on a platform-interpreted user input, perhaps including database operations. The application server then delivers to the platform an output page consisting of HTML and platform-specific markup, including scripts. The platform then interprets this output page, replacing the platform-specific markup with standard HTML and JavaScript, and delivers the interpreted output page to end users. A cryptographic module based on ActiveX is used to implement the decryption of output page in the client's browser.

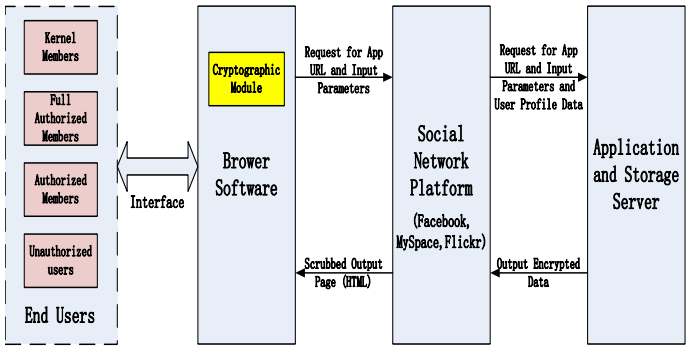


Figure 2: The application dataFlow for our architecture.

In this architecture, the resource publisher enforces access control through encryption and key management on our GCC scheme. Based on the above application dataflow, in Fig. 3 we describe a flowchart for publishing and accessing resources as follows:

- In a social network each user can choose a favorite label and generate a private key by herself, and then register her label into an OSN platform by *UserRegister* algorithm;
- When somebody wants to share resources with others, she constructs a community together with a set of trusted friends on an OSN platform by *BuildCommunity* algorithm. Finally, each member gets a community key, which can be used to access, manage and maintain the resources in this community;
- When a user wishes to access a community, her friends hold the community key can delegate an access permission key (APK) to her by using *DelegatePermission* algorithm;
- If one community member wants to post message and resource into the community, she picks the community key, invokes *UploadResource* algorithm to encrypt the resource with her private key, and then transmits the encrypted data to the storage server; and
- Anytime one community member can obtain the encrypted data from the sever, and invoke *Download-Resource* algorithm to retrieve the original post or resource by her private key and APK.

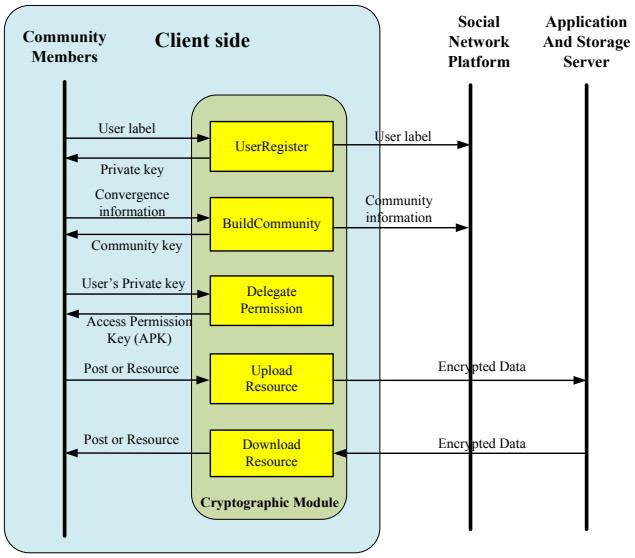


Figure 3: Cryptographic module and application flow for our architecture.

According to our description, we enforce access control and key management at the client side by a group of kernel members. In our architecture, we do not need to assume that the system manager is trusted to manage a private OSN, so that the community can be constructed in an autonomous and collaborative way, without the involvement of a system manager. To enable access control through key management without a system manager, our design should satisfy several important security and performance requirements, such as autonomy, independence, collaboration, authentication, and revocation.

4 Community Key Management

In this section, we articulate our scheme for community key management based on above-mentioned architecture. To design this scheme, our work addresses following problems: how do the kernel members define a community? how do the authorized members generate and distribute the community keys? how do the members grant access permissions corresponding to a community? how does an untrusted third party (e.g. the OSN platform) can authenticate the kernel members of community?

Table 2: Notations and symbols used in paper.

Term	Definition
$u.id$	u's label
$u.sk$	u's private key
$u.pm$	u's permission
$u.pk$	u's Elgamal public key
gk	a community key
Setup(κ)	Initiate the global parameter p of the system by a security parameter κ
Register(p, id)	Choose private keys in terms of global parameter p and id
Converge(S)	Generate convergence information Σ from the set of public keys of all kernel members S
CKeyGen($u.sk, \Sigma$)	Build the community key gk with a user's key $u.sk$ and a convergence information Σ
CEncrypt $(u.sk, u.pm, gk, F)$	Encrypt a resource F by using a user's key $u.sk$, a member's permission $u.pm$, and a community key gk
CDecrypt $(u.sk, u.pm, C)$	Decrypt the ciphertext C by using a user's key $u.sk$ and a member's permission $u.pm$
CVerify (F, C)	Verify the integrity of the resource F in the ciphertext C
Permission $(u.sk, gk, u'.id)$	Generate the access permission of a community by using a user's key $u.sk$, a community key gk and a target user's label $u'.id$
Revocation $(u.sk, gk, \mathcal{R}, F)$	Revoke a set of users $\mathcal{R}$ from a community by using a user's key $u.sk$ and a community key gk
EGSetup($u.sk, g$)	Generate ElGamal public key $u.pk$ from a user's key $u.sk$ and a generator g
EGEncrypt($u.pk, m$)	ElGamal encrypt message m to obtain a ciphertext c with a public key $u.pk$
EGDecrypt($u.sk, c$)	ElGamal decrypt ciphertext c with private key $u.sk$
KAuthenticate(A, B)	Authentication protocol between a kernel member A and a verifier B
FAuthenticate(A, B)	Authentication protocol between a (Full) authorized member A and a verifier B

In view of those problems, we propose a community key management scheme as follows: each user in OSN has an unique private key generated by *UserRegister* algorithm, while guaranteeing that OSN cannot know this key; community management mainly relies on three algorithms, *Buildcommunity*, *DelegatePermission* and *Revocation*, to build community and grant/revoke access permissions without the help of OSN; two algorithms, *UploadResource* and *DownloadResource*, are employed for creating, requesting, updating and deleting resources. In addition, a community maintains and enforces the public community's member list (CML). The kernel members may change the resource CML and revoke specific members to the resources by cryptographic revocation algorithm. Additionally, storage services in our model support two operations for data storage and retrieval: upload and download, which are re-realized by encryption and decryption operations on our GCC scheme. In short, the algorithms described in this section are able to allow different members to quickly and flexibly access data and resources in terms of their permissions.

Before we describe our construction, the symbols and notations in our GCC scheme are showed in Table 2. Detailed descriptions for these notations and corresponding algorithms are given in Appendix. We will make use of these symbols and notations to elaborate our construction.

4.1 UserRegister

First of all, the system manager invokes $Setup(\kappa)$ to generate a global parameter p and makes it public. Based on this parameter, any user u_i in OSN may choose a favorite label $u_i.id$ and generate his private key $u_i.sk$ by invoking $Register(id)$. Then, the manager registers this label after the user sends it to her.

Algorithm 1 UserRegister(κ):

1: manager: $p \leftarrow Setup(\kappa)$;
2: u_i: choose a favorite $u_i.id$;
3: u_i: $u_i.sk \leftarrow Register(u_i.id)$;
4: $u_i \rightarrow$ manager: $u_i.id$;

Note that, the manager only needs to execute $Setup(\kappa)$ one time and does not know the user's private key.

4.2 BuildCommunity

The BuildCommunity function allows a set of trusted users to build a community. In our scheme, a community is built by collaboration of a set of users, rather than defined by one user alone. Furthermore, the community key is obtained by convergence of information of these members, instead of specified by one user or the system manager.

For a set of trusted users $\mathcal{S} = \{u_1, \cdots, u_m\}$, anyone in $\mathcal{S}$, called the dealer, can build the community key gk as follows: the dealer chooses a random generator $g \in \mathbb{G}$ for this community, and distributes it to all users in $\mathcal{S}$; each user in $\mathcal{S}$ returns a temporary public key $u_i.pk$ (as the commitment of his private key) in terms of g for $i \in [1, m]$; next, the dealer generates a convergence information Σ from all temporary public keys $\{u_1.pk, \cdots, u_m.pk\}$, and then builds the community key gk in terms of $CKeyGen(u.sk, \Sigma)$ without the help of manager.

Algorithm 2 BuildCommunity($dealer, \mathcal{S}$):

1: dealer: $g \leftarrow Random(p)$; // to generate a random integer.
2: dealer $\rightarrow \mathcal{S}$: distribute g to all members in $\mathcal{S}$;
3: u_i: $u_i.pk \leftarrow EGSetup(u_i.sk, g)$;
4: $u_i \rightarrow$ dealer: $u_i.pk$;
5: dealer: $\Sigma \leftarrow$ Converge($S = \{u_1.pk, \cdots, u_m.pk\}$);
6: dealer: $gk \leftarrow$CKeyGen($u.sk, \Sigma$);
7: dealer $\rightarrow \mathcal{S}$: distribute gk to all members in $\mathcal{S}$;

4.3 DelegatePermission

The permission delegation is a process to transfer the permission of a member in the community to her friends. By DelegatePermission algorithm, the members delegate the "read" right of a community to the external users. In order to avoid a unbounded delegation, we require that only kernel members and full authorized members can employ this algorithm to delegate access permissions. This algorithm includes two steps: 1) the access permission pm is generated in terms of the user's label; and 2) the access permission pm is securely transmitted from the member to her friends. We make use of ElGamal encryption to build a secure channel.

Algorithm 3 DelegatePermission(u_i, u_j):

1: u_i: $g \leftarrow$ obtain from gk;
2: $u_i \rightarrow u_j$: g;
3: u_j: $u_j.pk \leftarrow EGSetup(u_j.sk, g)$;
4: $u_i \leftarrow u_j$: $u_j.pk$;
5: u_i: $u_j.pm \leftarrow Permission(u_i.sk, gk, u_j.id)$;
6: u_i: $c \leftarrow EGEncrypt(u_j.pk, u_j.pm)$;
7: $u_i \rightarrow u_j$: c;
8: u_j: $u_j.pm \leftarrow EGDecrypt(u_j.sk, c)$;

To assign the permission, the member u_i firstly retrieves the generator g in the community key gk and sends it to her friend u_j. On receiving g, u_j sets up a temporary ElGamal public key in terms of $EGSetup(u_j.sk, g)$ and returns the public key to u_i. And then u_i computes the access permission of u_j by his private key, the data received from u_j and the community key gk. Next, u_i encrypts the permission with u_j's temporary public key and sends it to u_j. Finally, u_j decrypts the ciphertext with her private key and recovers the access permission.

If the member u_i wishes to delegate the "write" right to her friend, she only needs to transmit the community key besides the permission pm. That is, u_i merely replaces the line 6 by $c \leftarrow EGEncrypt(u_j.pk, m_j.pm||gk)$, where $||$ denotes the concatenation operation for two strings.

4.4 UploadResource

The UploadResource function is a process that a kernel member or a full authorized member publishes a message for the community. Since the encryption is introduced, the member must hold a valid community key gk to implement this process. Thus, authorized members have no permission to publish messages. In addition, we propose an efficient authentication protocol – $FAthenticate(A, B)$ – to check the identification of members. This process can prevent illegal users to submit invalid ciphertexts to the community.

Algorithm 4 UploadResource(u_i, F):

1: $u \leftrightarrow SNP$: $b \leftarrow FAthenticate(u_i, SNP)$;
2: **if** b is true **then**
3: u_i: $C \leftarrow Encrypt(u_i.sk, u_i.pm, gk, F)$;
4: $u_i \rightarrow SNP$: C;
5: $SNP \rightarrow CSP$: upload(C);
6: **end if**

Suppose the member u_i wants to publish resource F for a special community G. Firstly, the u_i interacts with the social network platform (SNP) to verify whether she is an authorized member. After the u_i passes the authentication protocol, she can encrypt the message and then submit the ciphertext to SNP. Finally, the SNP uploads the ciphertext to a storage service provider (SSP).

4.5 DownloadResource

The DownloadResource function allows a member to access messages in a private OSN. In order to improve the performance, this function is executed on the cryptographic module of end user. By using the user's private key sk and the access permission pm for a certain community, any member u_i can decrypt encrypted resources obtained from the social network platform and the storage server according to the algorithm $Decrypt(u_i.sk, u_i.pm, C)$. Hence, all authorized members in a private OSN can retrieve encrypted data from the storage service provider.

Algorithm 5 AccessResources(u_i, C):

1: u_i: $F \leftarrow Decrypt(u_i.sk, u_i.pm, C)$
2: u_i: $b \leftarrow CVerify(F, C)$
3: **if** b is true **then**
4: u_i: Message is intact and output F
5: **end if**

In order to check the integrity of message, the GCC scheme provides an efficient verification algorithm $CVerify$ for the ciphertext by using the cryptographic Hash function. Hence, once the ciphertext has been decrypted, the member can verify whether the decrypted message is intact. If the result of this process is $true$, then the message can be returned to the Web browser.

4.6 Revocation

The Revocation function allows to exclude a set of members $\mathcal{R}$ from all authorized members. To avoid the disclosure of privacy, the revocation is an efficient mechanism to maintain the security of a private OSN during long-term running. With the help of revocation algorithm in the GCC scheme, we can implement the revocation as follows: given a set of revoked members $\mathcal{R}$ (obtained from the user's public label), the kernel or full authorized member can invoke the $Revocation(u_i.sk, gk, \mathcal{R}, F)$ to encrypt the message F by using the

private key and the community key. Such a revocation does not mean the authorized user will no longer access any resource in the group.

Algorithm 6 Revocation$(u_i, \mathcal{R}, F)$

1: u_i: $C \leftarrow$Revocation$(u_i.sk, gk, \mathcal{R}, F)$

If kernel members wish to revoke permanently an authorized member, she only needs to add this member into the revoked members list (RML) in the community, and then makes this RML public. While uploading the message into the community, it simply requires that the member uses this RML as the set $\mathcal{R}$ to encrypt the message.

Note that, in the GCC scheme the number of revoked users is strictly less than the number of kernel users in the group. In order to enhance the capacity of revocation, we can easily increase the number of revoked users by using some random keypair when the community key is generated.

5 Implementation and Application

5.1 Implementation of the GCC scheme

An experimental GCC cryptosystem was implemented to demonstrate the feasibility of our scheme. This system was developed with standard C++ language in QT environment, which supports cross-platform deployment. This system consists of three modules: Cryptographic Module, Private Social Network Platform, and Browser Software. In the cryptographic module, we adopted GNU multiple precision arithmetic library (GMP) to handle integers of arbitrary precision. Then, a finite field arithmetic library was constructed to realize the run-time environment of elliptic curve and pairing-based cryptosystems (in terms of PBC liberty from Stanford University). In addition, a Group-Oriented Convergence Cryptosystem library was developed based on the finite field arithmetic library to realize various GCC algorithms. Finally, the GCC algorithms worked with a lightweight private social network platform to provide encryption, authentication and key-label management services for Web browsers.

5.2 Application for a Blog management

We build a Blog management system where users are able to control access to her data without a third-party. This system supports the editing and publishing of blog posts, comments, and images. Posted data in this system are divided into two categories: public data that is visible to all users; and protected data that is visible only to the members of community that defined by the user. All Blog contents are stored at a server. The architecture of our application is represented in Fig. 4.

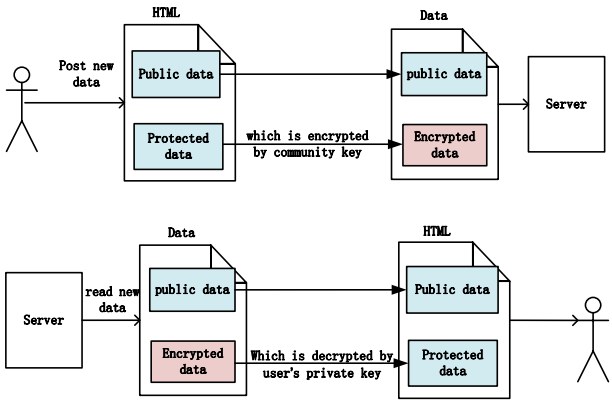
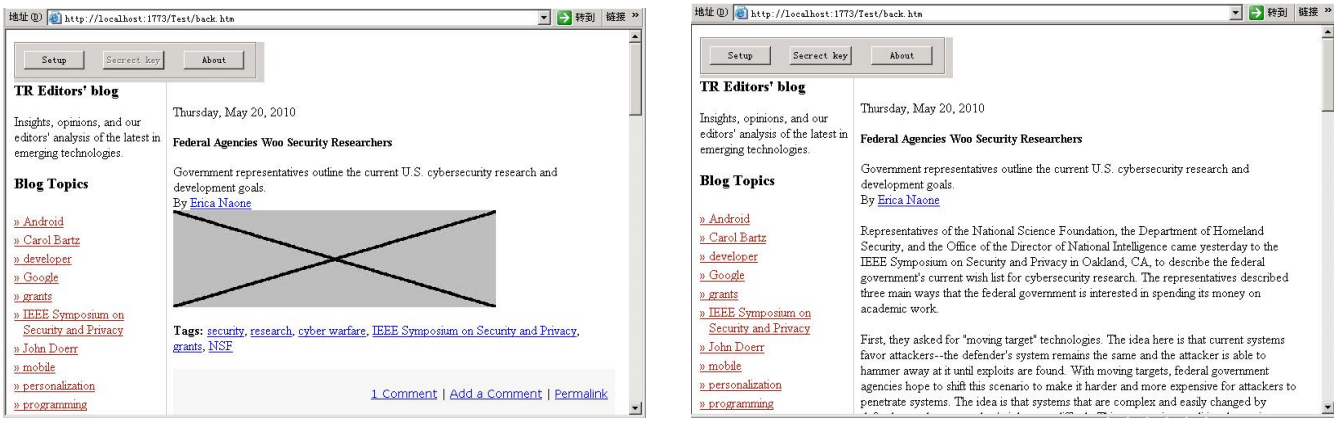


Figure 4: The Blog system architecture.

Once a user is about to post new data to her blog, she first decides which data is public and which data should be protected. For the protected data, she decides which members of community may have access to her data, then encrypts this data with her keys and corresponding community key. Public data together with

encrypted data are sent to the server. When somebody in the system browse user A's blog, she gets data from the server. The public data is directly displayed to her, while the protected data is displayed with a default page which means this data is meaningless to the visitor. To view the entire content, she first has to examine the header of ciphertext to check whether she has permissions to access the community. If she is an authorized user, she can decrypt the ciphertext and view all contents; Otherwise, the protected data are still unknown to her.



(a) An example of a private OSN before decryption. (b) An example of a private OSN after decryption.

Figure 5: An example of a private OSN before decryption (left) and after decryption (right).

Fig. 5 shows an example of our implementation. In this example, when a user downloads a HTML page from a private OSN, she can only see the public data and some gray frames which denote encrypted data (see Fig. 5 (a)). Note that, there exists an ActiveX control on the top of two sub-figures, which implements the functions of cryptographic module in Fig. 3. In order to display the encrypted data, the user must click the button on this ActiveX control and then input the user's private key and access permission for this community. If the key and the permission are valid for this community, the ActiveX control would decrypt the encrypted data and display them to the user. The result is showed in Fig. 5 (b).

6 Related Works

There has been a substantial amount of work addressing the problem of privacy protection in social networks. One area of research is to protect user's privacy by enforcing access control. For example, Carminati et al. [11] proposed an rule-based access control model which allowed users to specify access rules for their contents. An *access rule* consists of the resource identifier and a set of conditions which must be satisfied to be allowed to access the resource. A requestor is authorized to access an resource only if he provides the resource owner with a proof that she satisfies at least one of the corresponding access rules, by means of relationship certificates. This scheme enforces access control at client side. In addition, they proposed a mechanism to enforce access control for web-based social networks [13].

Besides protection of resources, some recent works address the privacy of relationships in social networks, since availability of information on relationships (trust level, relationship type) gives rise to security concerns: knowing who is trusted by a user and to what extent being trusted disclose a lot about user's thoughts and feelings. For example, Carminati et al. [4] described an access control model on relationship protection. In this model, the relationship certificates are encrypted using symmetric cryptographic algorithm and are treated as a resource: a certificate is granted only one satisfies a *distribution rule*, which is analogous to the *access rule*. Ferrer et al. [7] introduced a public-key protocol for private relationships, where certificates were encrypted asymmertrically and signed. But this scheme has drawbacks: relationship strengths are revealed to intermediate users and it requires multiple users to engage in a protocol for each new access. In [8], a similar scheme which also protects the relationship strengths was introduced.

Another way of enforcing access control is by means of key management. Some recent papers addressed this problem by using access hierarchy, which is considered as a social network graph. Atallah et al. [14] introduced a hierarchy key management scheme based on hash functions and CCA-secure encryption in terms

of relationship hierarchy on social networks. In this scheme, if there exists a path from node A to node B in the access graph, then A can derive B's key only using its own key and hash functions. In [15], a provably-secure time-bound hierarchical key assignment scheme was also proposed. Frikken et al. [10] introduced a novel scheme based on [14]. In their scheme, users specify access policies based on distance in the social network, and access control is enforced via key management. Moreover, this scheme is decentralized and does not require users to be online at the same time.

Some new cryptographic techniques were introduced into online social networks as well. In [12], a fine-grained access control scheme through attribute-based encryption (ABE) has been proposed. In this cryptosystem, the ciphertexts are labeled with sets of attributes, and the private keys are associated with access structures that control which ciphertexts a user is able to decrypt. Furthermore, in [3] a new OSN architecture (called as Persona) is also used to hide the user data with ABE, allowing users to apply fine-grained policies over who may view their data. This architecture achieves privacy by encrypting private contents and prevents misuse of a user's applications through authentication based on traditional public key cryptography (PKC).

Some recent works also focus on the area of privacy protection within some existing social network sites (SNS). For example, the solution in [16] offers the users' access control of their sharing data by hiding and mapping the selected information into a third-party storage. In [5], the personal information is encrypted with a pseudo-random substitution cipher from a public dictionary. However, this approach works only for encrypting personal data from a relatively small domain, and does not support encrypting free entries. In [17], a Facebook application called as flyByNight was presented to protect private data by storing it in encrypted form. By using traditional cryptosystem, the client-side encrypts the message with public-keys and decrypts ciphertext with private-key. This application places significant trust in the Facebook servers and relies on them to enforce key management.

7 Conclusions

In this paper, we introduced a scheme where resources are shared among communities, which means only members of a community have access to its resources. Adopting a community key management, we were able to keep users' resources confidential, even towards the system manager. In our framework, a random session key is used and encapsulated for each encryption, and only members can derive the session key and decrypt it correctly. Our proof-of-concept prototype clearly demonstrated that our scheme is practical to OSNs, allowing us to generate community keys with the manageable computation overhead.

Acknowledgments

This work of Gail-J. Ahn and Hongxin Hu was partially supported by the grants from National Science Foundation (NSF-IIS-0900970 and NSF-CNS-0831360). The work of Yan Zhu, Zexing Hu, and Huaixi Wang was partially supported by National Development and Reform Commission (under project "a monitoring platform for web safe browsing") and China Next Generation Internet CNGI Project (CNGI-09-01-12).

References

[1] W. Luo, Q. Xie, and U. Hengartner, "Facecloak: An architecture for user privacy on social networking sites," in *CSE (3)*. IEEE Computer Society, 2009, pp. 26–33.

[2] M. M. Lucas and N. Borisov, "flybynight: mitigating the privacy risks of social networking," in *SOUPS*, ser. ACM International Conference Proceeding Series, L. F. Cranor, Ed. ACM, 2009.

[3] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *SIGCOMM*, P. Rodriguez, E. W. Biersack, K. Papagiannaki, and L. Rizzo, Eds. ACM, 2009, pp. 135–146.

[4] B. Carminati, E. Ferrari, and A. Perego, "Private relationships in social networks," in *ICDE Workshops*. IEEE Computer Society, 2007, pp. 163–171.

[5] S. Guha, K. Tang, and P. Francis, "Noyb: privacy in online social networks," *Proceedings of the first workshop on online social networks*, pp. 49–54, 2008.

[6] B. Carminati and E. Ferrari, "Privacy-aware collaborative access control in web-based social networks," in *DBSec*, ser. Lecture Notes in Computer Science, V. Atluri, Ed., vol. 5094. Springer, 2008, pp. 81–96.

[7] J. Domingo-Ferrer, "A public-key protocol for social networks with private relationships," in *MDAI*, ser. Lecture Notes in Computer Science, V. Torra, Y. Narukawa, and Y. Yoshida, Eds., vol. 4617. Springer, 2007, pp. 373–379.

[8] J. Domingo-Ferrer, A. Viejo, F. Sebé, and Ú. González-Nicolás, "Privacy homomorphisms for social networks with private relationships," *Computer Networks*, vol. 52, no. 15, pp. 3007–3016, 2008.

[9] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati, "Preserving confidentiality of security policies in data outsourcing," in *WPES*, 2008, pp. 75–84.

[10] K. B. Frikken and P. Srinnivas, "Key allocation schemes for private social networks," in *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, 2009, pp. 11–20.

[11] B. Carminati, E. Ferrari, and A. Perego, "Rule-based access control for social networks," in *OTM Workshops (2)*, ser. Lecture Notes in Computer Science, R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 4278. Springer, 2006, pp. 1734–1744.

[12] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *ACM Conference on Computer and Communications Security*, 2006, pp. 89–98.

[13] B. Carminati, E. Ferrari, and A. Perego, "Enforcing access control in web-based social networks," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, 2009.

[14] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken, "Dynamic and efficient key management for access hierarchies," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 3, 2009.

[15] G. Ateniese, A. D. Santis, A. L. Ferrara, and B. Masucci, "Provably-secure time-bound hierarchical key assignment schemes," in *ACM Conference on Computer and Communications Security*, A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds. ACM, 2006, pp. 288–297.

[16] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman, "Lockr: Better privacy for social networks," in *In Proc. of the 5th coNEXT*, 2009.

[17] M. M. Lucas and N. Borisov, "Flybynight: mitigating the privacy risks of social networking," in *WPES*, V. Atluri and M. Winslett, Eds. ACM, 2008, pp. 1–8.

.1 Symbol lists

For the sake of clarity, we show the symbols used in the GCC scheme as follows:

.2 Definition of Group-oriented Convergence Cryptosystem

Definition 1 (Group-oriented Convergence Cryptosystem (GCC))**.** *A GCC scheme is a collection of polynomial-time algorithms* (Setup, Register, Converge, CKeyGen, CEncrypt, CDecrypt, CVerify, Permission, Revocation) *and two authentication protocols* (KAuthenticate, FAuthenticate) *such that :*

- Setup(κ) $\rightarrow \{p\}$ *is a probabilistic algorithm run by the social network manager to initiate the global parameter of the system. It takes as input a security parameter* κ*, outputs the system parameter* p*.*

Table 3: Notation and symbols in our scheme.

Notation	Meaning
$\mathcal{U}$	All the users in the social network
S	A subset of users
$\mathcal{R}$	A subset of revoked users
p	The global parameters of the system
pk, sk	The public and private keys of one user
gk	The community key
μ	Authorization information granted to user
M, V	Vandermonde matrix and its inverse matrix
$\mathbb{G}, \mathbb{G}_T$	Groups used in bilinear map
H_1	A secure hash function that maps strings to $\mathbb{G}$
H_2	A hash function that maps element in $\mathbb{G}_T$ to $\{0,1\}^n$
Σ	The convergence information

- Register$(p, id) \rightarrow \{sk\}$ *is an algorithm used to choose the private key. It takes as input the global parameter* p *and the* id*, outputs a private key* sk*.*

- Converge$(S) \rightarrow \{\Sigma\}$ *is an algorithm run by the dealer to generate convergence information. It takes as input a set of public keys of all kernel members* S*, and outputs the convergence information* Σ*.*

- CKeyGen$(sk, \Sigma) \rightarrow \{gk\}$ *is an algorithm run by dealer* u *to build a community. It takes as input* u*'s private key* sk*, and convergence information* Σ*. It outputs the community key* gk*.*

- CEncrypt$(sk, pm, gk, F) \rightarrow \{C\}$ *is a probabilistic algorithm run by the resource owner to securely publish his resource for a special community. It takes as input the private key* sk*, the permission* pm*, a community key* gk*, and the resource* F*. It outputs the valid ciphertext* C*.*

- CDecrypt$(sk, pm, C) \rightarrow \{F\}$ *is a deterministic algorithm used to decrypt the ciphertext. It takes as input the private key* sk*, the permission* pm*, the ciphertext* C*, and outputs the plaintext* F*.*

- CVerify$(C, F) \rightarrow \{true, false\}$ *is a deterministic algorithm run by the decipher to verify the integrity of resource* F *in the ciphertext* C*. It takes as input resource* F *and a ciphertext* C*. It outputs true if* F *is integrated , outputs false otherwise.*

- Permission$(sk, gk, id) \rightarrow \{pm\}$ *is an algorithm run by a community's member to grant reading right to user. Let* $\mathcal{C}$ *denote the community. It takes as input the private key* sk*, the community key* gk*, and the target user* v*'s id* id*. It outputs a permission* pm *which enables* v *to access resources in* $\mathcal{C}$ *but couldn't publish data for* $\mathcal{C}$*'s members.*

- Revocation$(sk, gk, \mathcal{R}, F) \rightarrow \{C\}$ *is an algorithm run by resource owner to publish resource, while a set of authorized users* $\mathcal{R}$ *are excluded from being able to decrypt the ciphertext. It takes as input the secret key* sk*, the community key* gk*, a subset of revoked user* $\mathcal{R}$*, and the resource* F*. It outputs the ciphertext which can not be decrypted by users in* $\mathcal{R}$*.*

- KAuthenticate$(A, B) \rightarrow \{true,false\}$ *is a protocol to verify whether* A *is kernel member of a community* $\mathcal{C}$*. If so,* A *can publish resource at* B *.*

- FAuthenticate$(A, B) \rightarrow \{true,false\}$ *is a protocol to verify whether* A *is full authorized user of a community* $\mathcal{C}$*. If so,* A *can publish resource at* B*.*

- Setup(κ): Given a security parameter $\kappa \in \mathbb{Z}^+$, the system manager does:
  1. Generate a random κ-bits prime q, two groups $\mathbb{G}, \mathbb{G}_T$ of order q, and an admissible bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathbb{T}$.
  2. Choose the symmetric encryption/decryption algorithm (E, D), two cryptographic hash functions $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \mathbb{G}_T \to \{0,1\}^n$, where n is the size of encryption key. The system parameter is $p = \langle q, \mathbb{G}, \mathbb{G}_T, e, n, H_1, H_2, E, D\rangle$.
- Register(p, id): Choose a random integer $y \in \mathbb{Z}_q^*$ as this user's the private key, while id should be an integer in $\mathbb{Z}_q^*$.
- Converge(S): Given a set of kernel members $S = \{u_1, \cdots, u_m\}$, the algorithm firstly collects the information $\Psi = \{(x_1, g^{y_1}), (x_2, g^{y_2}), \cdots, (x_m, g^{y_m})\}$ from S, then
  1. Suppose the underlying interpolation polynomial is $f(x) = a_0 + \sum_{i=1}^{m-1} a_i x^i$. Since $g^{y_k} = g^{f(x_k)}$, we get m equations: $g^{(a_0 + a_1 x_k + \cdots + a_{m-1} x_k^{m-1})} = g^{y_k}, k = 1 \ldots m$.
  2. Let $A = (a_0, a_1, a_2, \cdots, a_{m-1})^T$, $Y = (y_1, y_2, y_3, \cdots, y_m)^T$ be the unknown exponents, the Vandermonde matrix: $M = (m_{i,j}) = (x_i^{j-1}), 1 \le i, j \le m$. Compute the inverse of matrix M: $M^{-1} = (v_{i,j}), 1 \le i, j \le m$
  3. Since $MA = Y$, then $A = M^{-1}Y$, $g^{a_i} = \prod_{j=1}^{m} (g^{y_j})^{v_{i+1,j}}$. Output $\Sigma = \{g^s = g^{a_0}, g^{a_1}, \cdots, g^{a_{m-1}}\}$.
- CKeyGen(sk, Σ): Let private key be y, convergence information Σ be $\{g^s, g^{a_1}, \cdots, g^{a_{m-1}}\}$
  1. For $1 \le k \le m-1$: choose a random $l_k \in_R \mathbb{Z}_q^*$ that is not the same as previous, compute $g^{f(l_k)} = g^s \cdot \prod_{i=1}^{m-1} (g^{a_i})^{(l_k)^i}$.
  2. The community key is $gk = \{g, (l_1, g^{f(l_1)}), \cdots, (l_{m-1}, g^{f(l_{m-1})})\}$.
- CEncrypt(sk, pm, gk, F): Let private key be y, permission be μ, community key be $\{g, (l_1, g^{f(l_1)}), \cdots, (l_{m-1}, g^{f(l_{m-1})})\}$
  1. Compute $\rho = g^y$ if the encrypter is kernel user or $\rho = \mu$ if he is full authorized user. Recover $g^s = \rho^{\lambda_0} \prod_{i=1}^{m-1} (g^{f(l_i)})^{\lambda_i}$, where $\lambda_i = \prod_{0 \le k \le m-1, k \ne i} \frac{l_k}{l_k - l_i} \pmod q$, l_0 is encrypter's id.
  2. Choose a random integer $r \in \mathbb{Z}_q^*$, compute the hash of F: $H_1(F) \in \mathbb{G}$ and $g^{s_2 r} = (H_1(F)/g^s)^r$.
  3. Choose a random session key $ek \in_R \mathbb{G}_T$, compute the header of ciphertext
$$hdr = \langle g, g^r, g^{s_2 r}, ek \cdot e(H_1(F), g^r), (l_1, e(g^r, g^{f(l_1)})), \cdots, (l_{m-1}, e(g^r, g^{f(l_{m-1})}))\rangle.$$
  4. Compute the symmetric key: $K = H_2(ek)$, encrypt it symmetrically, i.e. $C_F = E(K, F)$. Output the ciphertext $C = (hdr, C_F)$.
- CDecrypt(sk, pm, C): Let private key be y, permission be μ, and ciphertext header be $hdr = \langle g, c_1, c_2, c_3, (l_1, \tau_1), \cdots, (l_{m-1}, \tau_{m-1})\rangle$
  1. Compute $\rho = g^y$ if the encrypter is kernel user or $\rho = \mu$ if he is full authorized user. Recover the session key ek as follows: $ek = \frac{c_3}{e(c_1, \rho)^{\lambda_0} \cdot \prod_{i=1}^{m-1} (\tau_i)^{\lambda_i} \cdot e(g, c_2)}$ where $\lambda_i = \prod_{0 \le k \le m-1, k \ne i} \frac{l_k}{l_k - l_i} \pmod q$, l_0 is the decipher's id.
  2. Recover the symmetric decryption key $K = H_2(ek)$, decrypt ciphertext body with K, i.e. $F = D(K, C_F)$ where D is the decryption algorithm and C_F is the ciphertext body.
- CVerify(F, C): Let ciphertext header of C be $hdr = \langle g, c_1, c_2, c_3, (l_1, \tau_1), \cdots, (l_{m-1}, \tau_{m-1})\rangle$. Compute ek as described in CDecrypt algorithm, then compute hash value $H_1(F)$ and $\varsigma = ek \cdot e(H_1(F), c_1)$. If $\varsigma = c_3$, then output "true". Output "false" otherwise.
- Permission(sk, gk, id): Let private key be y, community key be $\{g, (l_1, g^{f(l_1)}), \cdots, (l_{m-1}, g^{f(l_{m-1})})\}$ and target user's id be x_1.
  1. Obtain $\Sigma = \{g^s, g^{a_1}, \cdots, g^{a_{m-1}}\}$ by calling $Converge((x, g^y), (l_1, g^{f(l_1)}), \cdots, (l_{m-1}, g^{f(l_{m-1})})$, x is delegator's id.
  2. Compute $g^{f(x_1)} = g^s \cdot \prod_{i=1}^{m-1} (g^{a_i})^{x_1^i}$, outputs the permission $pm = g^{f(x_1)}$.
- Revocation($sk, gk, \mathcal{R}, F$): Let private key be y, community key be $\{g, (l_1, g^{f(l_1)}), \cdots, (l_{m-1}, g^{f(l_{m-1})})\}$, and the revoked users be $\{u_1, u_2, \cdots, u_t\}$. This algorithm has five steps, but steps $1, 2, 5$ are the same as steps $1, 2, 4$ in *CEncrypt* algorithm separatively. We describe steps $3, 4$ as follows:
  3. Choose a random session key $ek \in_R \mathbb{G}_T$ and call $Converge((x, g^y), (l_1, g^{f(l_1)}), \cdots, (l_{m-1}, g^{f(l_{m-1})})$ algorithm to obtain $\Sigma = \{g^s, g^{a_1}, \cdots, g^{a_{m-1}}\}$
  4. Suppose the public keys of revoked users are $\{x_1, x_2, \cdots, x_t\}$, compute the header of ciphertext as:
$$hdr = \langle g, g^r, g^{s_2 r}, ek \cdot e(H(F), g^r), (l_1, \tau_1), \cdots, (l_{m-t-1}, \tau_{m-1-t}), (x_1, \tau_{m-t}), \cdots, (x_t, \tau_{m-1})\rangle.$$
  where $\tau_i = e(g^r, g^{f(l_i)})$, for $1 \le i \le m-t-1$; $\tau_i = e(g^r, g^{f(x_i)})$,$g^{f(x_i)} = g^s \cdot \prod_{k=1}^{m-1} (g^{a_k})^{x_i^k}$, for $m-t \le i \le m-1$.
- KAuthenticate(A, B): Let A's id and private key be x, y, the public information B holds is $\{e(g,g)^s, (l_1, e(g,g)^{f(l_1)}), \cdots, (l_{m-1}, e(g,g)^{f(l_{m-1})})\}$:
  1. B chooses a random integer $t \in \mathbb{Z}_q^*$ and sends it to A
  2. A chooses a random integer $r \in \mathbb{Z}_q^*$, computes $\omega = r + yt$ and $\phi = e(g,g)^r$, then sends $\{\omega, \phi, x\}$ to B
  3. B computes $\upsilon = (e(g,g)^{\omega \lambda_0} / \phi^{(\lambda_0 - 1)}) \prod_{i=1}^{m-1} [e(g,g)^{f(l_i)}]^{\lambda_i t}$, $\lambda_i = \prod_{0 \le k \le m-1, k \ne i} \frac{l_k}{l_k - l_i} \pmod q$ for $\{l_0 = x, l_1, \cdots, l_{m-1}\}$.
  4. If $\upsilon = \phi \cdot [e(g,g)^s]^t$, then B outputs "success". Otherwise B outputs "failure".
- FAuthenticate(A, B): Let A's id be x, A's permission be μ, A's community key be $\{g, (l_1, g^{f(l_1)}), \cdots, (l_{m-1}, g^{f(l_{m-1})})\}$, and the public information B holds is $\{e(g,g)^s, (l_1, e(g,g)^{f(l_1)}), \cdots, (l_{m-1}, e(g,g)^{f(l_{m-1})})\}$:
  1. B chooses two random integer $\{x_1, t \in \mathbb{Z}_q^*\}$ and sends them to A
  2. A chooses a random integer $r \in \mathbb{Z}_q^*$, computes $\phi = e(g,g)^r$ and obtains $\Sigma = \{g^s, g^{a_1}, \cdots, g^{a_{m-1}}\}$ by calling $Converge((x, \mu), (l_1, g^{f(l_1)}), \cdots, (l_{m-1}, g^{f(l_{m-1})})$.
  3. A computes $g^{f(x_1)} = g^s \cdot \prod_{i=1}^{m-1} (g^{a_i})^{x_1^i}$ and $\omega = e(g,g)^r \cdot [e(g,g)^{f(x_1)}]^t$, then sends $\{\omega, \phi\}$ to B.
  4. B computes $\upsilon = (\omega^{\lambda_0} / \phi^{\lambda_0 - 1}) \cdot \prod_{i=1}^{m-1} (e(g,g)^{f(l_i)})^{\lambda_i t}$, where $\lambda_i = \prod_{0 \le k \le m-1, k \ne i} \frac{l_k}{l_k - l_i} \pmod q$ for $\{l_0 = x_1, l_1, \cdots, l_{m-1}\}$.
  5. If $\upsilon = \phi \cdot [e(g,g)^s]^t$, then B outputs "success". Otherwise B outputs "failure".