P2P Social Networks With Broadcast Encryption Protected Privacy

Oleksandr Bodriagov, Sonja Buchegger

School of Computer Science and Communication
KTH - The Royal Institute of Technology
Stockholm, Sweden
obo@kth.se buc@csc.kth.se

Abstract. Users of centralized online social networks (OSN) do not have full control over their data. The OSN provider can collect and mine user data and intentionally or accidentally leak it to third parties. Peer-to-peer (P2P) social networks address this problem by getting rid of the central provider and giving control to the users. However, existing proposals of P2P social networks have many drawbacks: reliance on trust, expensive anonymization or encryption techniques, etc.
We propose to use broadcast encryption for data protection because of its efficiency and ability to not disclose information about who can decrypt what. We present an architecture of a P2P social network that uses a composition of public-key cryptography, broadcast encryption, and symmetric cryptography. The architecture provides confidentiality and limited integrity protection. It defines privacy-preserving profiles that allow users to quickly find data encrypted for them while preventing attackers from learning who can access which data.

Keywords: P2P Social Network, Provider-independent, Encryption-based access control, Broadcast Encryption

1 Introduction

Existing centralized, provider-dependent networks do not provide users with mechanisms to fully protect their data. The provider has complete control of the service, and users have to rely on security mechanisms provided by the service. There is no guarantee that the trusted provider will enforce the privacy preferences of the users. Besides, there can be data mining, targeted advertisements, and even disclosure of users data to third parties.

Our aim is to give full control of the data to the users in order to prevent any misuse by third parties. While laws provide an important deterrent, relying on legal protection (or, alternatively on trust relationships) is not enough to prevent a determined misbehaving third party or social networking service provider from getting access to the data and using it in some way. Laws and regulations give incentives for appropriate treatment of personal data in terms of a potential post factum punishment to follow the rules. Cryptographic mechanisms, on the other

hand, can prevent unwanted access to data in the first place by technical means. Therefore, the PeerSoN project[1] was started to design a peer-to-peer (P2P) provider-independent architecture with cryptographically protected privacy.

One of the main problems of P2P social network architectures is to achieve secure, efficient, 24/7 access control enforcement and data storage. None of the current P2P architectures for social networks manages to fully cope with this problem. For example, Safebook [1] relies on trusted peers for data storage and profile data retrieval, and uses an expensive anonymization technique based on asymmetric cryptography; Diaspora [2] has quite high expectations on the users' willingness to run their own servers or have their data served and stored on servers of other users (a fully P2P variant of Diaspora requires users to run and manage their own servers); Persona [3] relies on a ciphertext-policy attribute-based encryption (CP-ABE), and current ABE schemes are very computationally intensive and produce ciphertexts that are linear in the number of attributes (too expensive for the P2P storage). Besides, CP-ABE schemes by definition support only open access structures, which can be considered as a security flaw since it is easy to infer who can decrypt what using that information.

In complex systems such as social networks, with many subjects and objects, with fine-grained access control (with objects encrypted differently and separately), the efficiency of encryption/decryption schemes is very important as usability depends on it. To achieve an efficient encryption-based access control with high performance encryption and decryption regardless of the number of identities/groups we use broadcast encryption (BE) schemes. The storage is assumed to be a P2P untrusted storage with multiple replicas, so that data is stored on a profile owner's computer/mobile phone (primary copy) and on chosen peers (replicas). The data is encrypted and everyone can download it, i.e. access control is encryption based. Even if the content is encrypted, there are consequent privacy implications of inferences from traffic analysis such as access patterns or properties of the stored files. Therefore, the architecture includes mechanisms to mitigate this threat.

Both the encrypted data and the broadcast encryption private keys for decrypting the data are stored at the profile owner's storage. These private keys are encrypted under public keys of intended recipients. Besides encrypted data and the BE keys, there is also public information (unencrypted) that allows others to find a user's profile. Broadcast encryption is used for data dissemination to groups and public key cryptography is used for user-to-user messaging. A multicast messaging (one message for several users) is realized via BE.

The rest of the paper is organized as follows. At first we explain the broadcast encryption, then we describe which changes we made to the chosen BE scheme and how it works. We continue by describing the architecture of the system in more detail. Then we discuss the security issues and define an attacker model. We finish by drawing the conclusions.

[1] http://www.peerson.net/

2 Broadcast encryption

Broadcast encryption (BE) schemes are used to distribute encrypted data to a dynamic set of users in a cost-effective way. In general, BE scheme consists of a sender and a group of recipients. Each recipient has his/her own private decryption key to decrypt encrypted data sent by the sender.

BE schemes can either be symmetric or public key based. In the first case, only a trusted source/broadcaster of the system that generated all the private keys can broadcast data to receivers. If the system is public key based, then anyone who knows a public key of the system can broadcast.

The efficiency of BE schemes is measured in terms of transmission cost, user storage cost, and computational cost. Besides efficiency, one of the main requirements for BE schemes is that it should be easy to revoke a key/group of keys. Other important security concepts are collusion resistance and statelessness. A fully collusion-resistant scheme is robust against collusion of any number of revoked users. A BE scheme is said to be stateless if after revocation of some subset of users the remaining users do not have to update their initial private keys. A BE is called dynamic [4] if new users can join without a need to modify existing users' decryption keys, if the ciphertext size and the system's initial key setup do not depend on the number of users, if the group public key should be incrementally updated with complexity at most $O(1)$.

Suitable candidates for application to a social network scenario are BE schemes with the following properties: stateless, fully collision resistant, with hidden set of receivers, dynamic, with constant size ciphertexts and keys, with computationally efficient decryption.

We use a dynamic identity-based broadcast encryption (DIBBE) scheme that meets all these requirements [5]. Although identity-based schemes involve a third-party authority - a Private Key Generator (PKG), this role is given to the profile owner when adjusting this IBBE scheme for the social network scenario. Thus, the profile owner is responsible for creating a group of receivers and assigning private BE keys to receivers. The IBBE scheme is formaly defined as a tuple of algorithms *IBBE = (Setup, Extract, Encrypt, Decrypt)* [6]. Although the DIBBE scheme defined in [5] has the same structure, there are some differences in the algorithms' input parameters that reflect a dynamic nature of the scheme. The algorithms of the used DIBBE scheme have the following form:

Setup(λ)$\rightarrow$ (MK, GPK): This algorithm takes as input a security parameter λ and constructs a secret master key *MK*, a group public key *GPK*.

Extract(MK, Id)$\rightarrow$ Sk_{Id}: A user's private key Sk_{Id} is generated by this algorithm that takes as input a user Id and a secret master key *MK* known only to the profile owner.

Encrypt(S, MK, GPK)$\rightarrow$ (Header,K): The used scheme is constructed as a Key Encapsulation Mechanism (KEM) which means that the encryption algorithm *Encrypt* takes as input the set of receivers *S*, the master key *MK*, and the group public key *GPK* and outputs a pair *(Header,K)*, where *K* is a symmetric secret key to encrypt data and *Header* is an encryption of this symmetric key for the set of receivers *S*. Data is stored in the form *(Header, encrypted data)*, and

Header reveals no information about the set of receivers or any other parameters. Only a user whose ID is in the set can decrypt the *Header* using his/her private key. Users that are not members of the group cannot encrypt to the group even if they know *GPK*, because the master key *MK* is required for the encryption process.

Decrypt(Id, GPK, Sk_{Id}*, Header)*$\rightarrow$ *K:* The *Decrypt* algorithm takes *GPK*, *Header*, the private key, and the user Id as input and outputs a symmetric key *K*.

Revocation of users from a group is a simple though computationally intensive operation. Revocation of the group membership for stateless BE schemes does not require re-keying for other group members, only re-encryption of the data with a new symmetric key and consequent regeneration of *Headers* for the new set of receivers. Addition of a user to a group in dynamic schemes requires re-encryption of Headers for the new set of receivers in addition to creating a private key for a new user.

3 Architecture: privacy preserving profiles

All objects in the profile that are not public are encrypted using a symmetric cipher and stored without any kind of header. The links that lead to these objects are encrypted using the broadcast encryption and stored in the form *(Header, encrypted link)*. To access the data the user has to decrypt the *Header* and get the symmetric key, use this key to decrypt the link, follow the link, and decrypt the data using the same key. The *Header* contains an implicit access control list (ACL) with identities of those who can decrypt it, and a user cannot know whether he/she can decrypt the object without trying. Therefore, for performance reasons users should be able to determine whether they can decrypt particular object without actually trying to decrypt it. At the same time, one of the aspects of privacy is to prevent other users from learning who can access which objects. Therefore, in the proposed architecture explicit ACLs are not stored alongside encrypted objects. Instead the privacy preserving profile contains for each of the contacts a folder with values that represent the BE groups in which that contact is a member and, possibly, links to encrypted objects that can be decrypted by that contact.

The links folder is encrypted under a shared symmetric key known only to the user and the profile owner. The links folders have random identifiers in order to prevent anyone from seeing all profile owner's connections. Any two users that want to form a connection should explicitly state to each other an identifier for the proper links folder during connection establishment. Identifiers of the links folders should be changed on the regular basis. The new identifier can be securely communicated to the contact in multiple ways. To hide the total number of contacts from other users, a profile contains a set of dummy links folders that are also updated from time to time. A padding should be used to make dummy folders indistinguishable from the real. To hide a real number of objects in the

profile dummy objects are created and updated on the regular basis. Thus, no user can calculate which percentage of objects he/she can access.

Figure 1 depicts the described user profile. The records that have the same colour of the key are encrypted under the same encryption key. The storage stores all profile data.

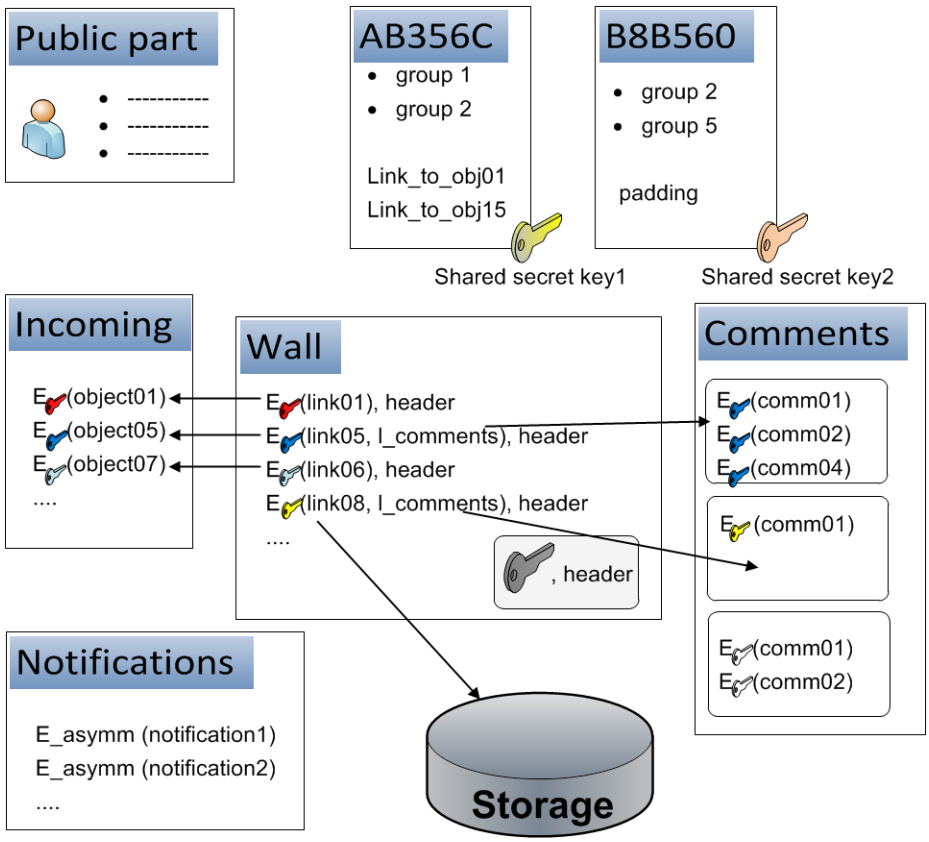

Fig. 1. System architecture: user profile

The privacy preserving profile consists of special purpose folders *Incoming*, *Notifications* and general purpose folders for posting (e.g. wall) to which other users can only add messages, comments folder, links folders for the contacts, and other folders that cannot be modified by other users. The comments folder contains subfolders with comments for each of the objects with allowed commenting.

The general purpose folder contains a list of BE encrypted links. These links are encrypted for different sets of receivers and their *Headers* contain values that represent the BE groups (the set of receivers) that can decrypt these links.

Users that can decrypt the links can get access to the data. Along with a link to the object itself, there can be another link that leads to a comments folder for this object. So, if the user decrypts the links file and finds also the comments link, then he/she can create comments in that comments folder using the same symmetric key obtained after decrypting the *Header*.

4 Architecture: operations

The properties of the encryption function of the used BE scheme make it more efficient to encrypt a message for one big group of users than for several small. Therefore, all contacts should be put into one BE group. Since there is only one broadcast group per profile, each contact receives one private BE key. The profile owner creates also one public-private key pair for the group and gives each user the private key for authenticating to replica holders. Replica holders are assumed to know the public key of the public-private key pair for the group and also the public key of the profile owner.

The division of users into security groups is abstract: each abstract security group corresponds to some set of receivers S that can be easily modified during encryption. This flexibility allows us to create as many groups as required without affecting efficiency or manageability. Besides, only one parameter in the *Header* depends on the set of receivers, and it is always the same for the same set of receivers. Thus it can be cached to increase encryption speed. This parameter is put into links folders of the security group members. It is used to search general purpose folders for decryptable items.

The profile owner is the only entity that determines the access rights to the encrypted links and, consequently, objects. The other users when they want to post something in some general purpose folder should decrypt the symmetric key used for making posts in that folder. Access rights are defined when the profile owner BE encrypts this key for a particular set of receivers.

The profile owner's computer/mobile phone has an overlay security system for the local profile that reduces complexity of rights management using abstract groups instead of a list of identities. This local profile can store ACLs with abstract groups and separate IDs alongside with objects. The overlay security system is responsible for authentication and authorization of users in case of direct end-to-end communication.

The architecture contains the following operations:

Establish connection: A user creates a new links folder and runs a *Extract* algorithm to generate a BE private key for the new connection. The BE key, as well as the private key for the group, is encrypted under the public key of the new connection and is put in the newly created links folder. Then the two users exchange links folder identifiers and symmetric keys to decrypt these folders. All communication in this phase goes through the encrypted channel.

Add to a group: A user adds the ID of the new connection to a list of identities of some abstract security group and recalculates a value that depends on the set of identities in the *Header*. Then the profile owner adds this value to

the links folder of the new member and updates this value for all other members of the group.

Publish data(by the profile owner): The owner defines a set of contacts who will have access to the published data. Then he/she runs the broadcast encryption algorithm *Encrypt* that outputs a symmetric key to encrypt the link to the data and the data, and the *Header* (encryption of this key for the defined set of contacts). The profile owner encrypts the data with the generated symmetric key and stores it in the folder for encrypted objects, the link is also encrypted and stored along with the *Header* in some general purpose folder (e.g. Wall). Before going off-line the profile owner should synchronize all the changes with replicas.

Publish data(by a contact): If the profile owner's computer/mobile phone is online, then a contact communicates directly with the overlay security system of the profile owner via protected channel. If the security system determines that the contact has rights to publish data, it accepts the data, stores it on the local storage, and starts information update mechanism with peers. In case the profile owner is off-line, the steps are as follows. General purpose folders like Wall contain BE encrypted links and BE encrypted symmetric keys for posting in those folders. The encryption of the links in the general purpose folder determines who can read/comment on data. When the user who has rights to access the general purpose folder wants to add some message, he/she creates a new message and encrypts it with the symmetric key that is used for this folder, uploads this message to the *Incoming* folder of the profile, encrypts a link to the newly created message with the folder's symmetric key, signs it using the group private key, and adds this encrypted link to the general purpose folder. Then he adds an encrypted notification which informs the profile owner about the new message to the *Notifications* folder. The replicas are supposed to keep the integrity of the special purpose folders *Incoming* and *Notifications*, and general purpose folders by allowing only additions. Any message sent to replicas by any user except the owner should be signed by the group private key and the signature should be checked by the replica.

Comment(by a contact): If the profile owner's computer/mobile phone is online, then a contact communicates directly with the overlay security system of the profile owner via protected channel. If the security system determines that the contact has rights to comment on a particular object, it accepts the comment, stores it on the local storage, and starts information update mechanism with peers. In case the profile owner is off-line, the steps are as follows. General purpose folders like wall contain BE encrypted links and BE encrypted symmetric keys for posting in those folders. The encryption of the links in the general purpose folder determines who can read/comment on data. When the user who has rights to comment to some object wants to add a comment, he/she creates a new comment and encrypts it with the symmetric key that is used to encrypt the object and the link to the object, follows the link to the folder where the comments for this object are stored, signs the comment using the group private key, and uploads this comment to the comments folder of this object. Then

he/she adds an encrypted notification which informs the profile owner about the new comment to the *Notifications* folder. The replicas are supposed to keep the integrity of the comments folders by allowing only additions. Any message sent to replicas by any user except the owner should be signed by the group private key and the signature should be checked by the replica.

Send notification: Notifications are sent for messages published by the sender on the receivers profile and for the information published on the sender's profile. Notifications are encrypted under the public key of the receiver and signed with the group private key and uploaded to the *Notifications* folder of the receiver's profile if the receiver is off-line. When the receiver is on-line, the notification is sent directly to the receiver.

A contact can send the profile owner a notification asking to delete that user's post/comment. Only the profile owner should be able to do it, replicas are supposed to protect integrity of the profile from everyone else by allowing only additions.

5 Security considerations

The owner's local computer stores the primary copy of the profile and replica holders store only copies with limited modification possibility. Replica holders are assumed to enforce an "only addition" policy for the special purpose folders *Incoming* and *Notifications* and the general purpose folders, and the "no modification" policy for the rest of the folders for any user other than the owner. A user (other than the owner) can upload encrypted links to the general purpose folders (comments to the comments folder) and add messages to the special purpose folder only if they contain correct group signatures. The group signature check does not reveal any identities and protects from resource exhaustion attacks from external entities. Even if the general purpose folder is modified in some way by a malicious user, it can be easily recovered via notifications.

The profile owner keeps the primary copy of the profile and synchronizes it with replicas. We assume that any peer that takes part in the social network can be a replica. Communication between the profile owner and replicas goes through a protected channel. The channel is encrypted and authenticated using public-key authentication schemes.

We define a capability-based model of an attacker for the system as following. The attacker is an active entity, external to the system (does not receive any keys from the profile owner), that can direct attacks against the replica storage, the local computer/mobile phone of the profile owner and the computers of profile owner's friends. The attacker cannot perform computationally infeasible calculations in the reasonable time. The attacker can sniff and tamper all communication channels and send arbitrary messages to all participants.

If the attacker can compromise the primary copy of the profile (e.g. by breaking the security of the operating system or the overlay security system), then the attacker has full control of this profile. It is plausible that in case of such an attack the attacker learns private keys given by other profile owners to the

compromised owner. Then the attacker would be able to completely impersonate the user, read data intended for this profile owner from the profiles of other users, and would be able to exhaust replica's resources. If replicas keep integrity of the folders, the attacker will not be able to delete or modify old messages.

Secret keys used in encryption are prone to ageing. Re-encryption of data with a new underlying symmetric key is a relatively straightforward operation, but may be time consuming for big amounts of data. If each new encrypted file is encrypted with a separate symmetric key, then symmetric keys do not age. When a user publishes data on someone's else wall, he has to decrypt the symmetric key used for making posts in that folder first, and then use it to encrypt the data. This symmetric key should be changed regularly to prevent key ageing. However, if symmetric keys are changed very often, then the secret master key MK, which is used by the profile owner for encryption (to generate a symmetric key, see Section 2), will age very fast and will require frequent refreshments. Thus there is a trade-off between ageing of symmetric keys and the master key. The key refreshment issue is very important, but we do not address it in this work.

6 Conclusions

A P2P provider-independent architecture with cryptographically protected privacy is a straightforward solution to give full control over their data to the end-users and guarantee its protection.

Although broadcast encryption schemes were intended for a multi-recipient broadcasting, their properties make them suitable candidates for application to a social network scenario. We evaluated existing schemes for the suitability looking at the several criteria (efficiency, recipient privacy, etc.) and defined properties that are crucial for the BE schemes to be used in the social network scenario. We found one BE scheme that meets all the requirements, transformed it from the Identity-based BE scheme to the ordinary BE scheme, and adapted to the social network scenario.

With efficiency in mind, we proposed a P2P social network architecture that uses a composition of public-key cryptography, broadcast encryption schemes, and symmetric cryptography. The architecture provides confidentiality and limited integrity protection from an external attacker. The architecture defines a privacy preserving profile that allows users to quickly find data encrypted for them while preventing both the external attackers and malicious insiders from learning who can access which data.

Acknowledgments

This research has been funded by the Swedish Foundation for Strategic Research grant SSF FFL09-0086 and the Swedish Research Council grant VR 2009-3793.

References

1. Cutillo, L., Molva, R., Strufe, T.: Safebook: A privacy-preserving online social network leveraging on real-life trust. Communications Magazine, IEEE **47**(12) (2009) 94 –101
2. Grippi, D., Sofaer, R., Salzberg, M., Zhitomirsky, I.: Diaspora. a little more about the project (April 2010)
3. Baden, R., Bender, A., Spring, N., Bhattacharjee, B., Starin, D.: Persona: an online social network with user-defined privacy. SIGCOMM Comput. Commun. Rev. **39** (August 2009) 135–146
4. Delerablee, C., Paillier, P., Pointcheval, D.: Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In: Pairing-Based Cryptography Pairing 2007. Volume 4575 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2007) 39–59
5. Jiang, H., Xu, Q., Shang, J.: An efficient dynamic identity-based broadcast encryption scheme. In: Data, Privacy and E-Commerce (ISDPE), 2010 Second International Symposium on. (2010) 27 –32
6. Delerablee, C.: Identity-based broadcast encryption with constant size ciphertexts and private keys. In: Advances in Cryptology ASIACRYPT 2007. Volume 4833 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2007) 200–215