Practical Security in P2P-based Social Networks

Kalman Graffi, Patrick Mukherjee, Burkhard Menges, Daniel Hartung, Aleksandra Kovacevic, and Ralf Steinmetz
Multimedia Communications Lab[1] and Real-Time Systems Lab[2], Technische Universität Darmstadt, Germany
The Norwegian Information Security Laboratory[3], Gjøvik University College, Norway
Email: {graffi[1],sandra[1],steinmetz}@kom.tu-darmstadt.de, mukherjee[2]@es.tu-darmstadt.de, daniel.hartung[3]@hig.no

Abstract— **The peer-to-peer paradigm is used in more and more advanced applications. One of the next areas that promise a success for the p2p paradigm lies in the upcoming trend of social networks. However, several security issues have to be solved in p2p-based social network platforms. We present in this paper a practical solution that establishes a trust infrastructure, enables authenticated and secure communication between users in the social network and provides personalized, fine grained data access control. We implemented our solution in a p2p based platform for social networks and show that the solution is practical and lightweight both in time consumption and traffic overhead.**

I. Introduction

Social networking sites are web-based platforms allowing users to publish personal profiles, link each other, post pictures, blog entries, join groups and search for friends. Several hundred millions of users participate in today's social networks like Facebook or MySpace. However, due to the centralized character of this platforms, high server maintenance cost exists. A p2p-based approach solves the load and cost issues but leads to new challenging security issues for secure communication and data access. In this paper we present a practical approach for solving these security issues and give an evaluation regarding the costs of our solution.

Social networks provide a wide set of functionality, enabling users to publish and comment private profile pages, create photo albums, join and manage (interest) groups, search for users and groups and communicate with friends and groups through a messaging system. We summarize briefly our approach for a p2p-based platform for social networks, which can be found in [2]. We split the wide set of functionality into individual modular functionality blocks, like friends management, photo management and propose a plugin based architecture. Plugins can communicate with each other, so that more advanced plugins can be created by reusing the functionality of existing plugins as shown in Figure 1(a). Plugins operate also on an Information Cache which manages data storage and retrieval and also quickens the access to previously requested data. The Data storage is completely decentralized through the usage of a structured p2p overlay and a corresponding Storage and Replication layer. It provides the functionality of a distributed hash table (DHT) and ID-based routing, compliant to the Key-Based Routing (KBR) interface proposed by Dabek et al. in [3]. Typical data structures in social networks are lists. Friends lists, group membership lists, album lists and photo lists are examples for this data structure. We propose in [2] a distributed data structure based on lists, as seen in Figure 1(b). All storable data structures are reduced to storable list elements containing meta data and pointers to other lists or storable objects (SharedItems) which are identifiable by their unique object ID. Using a DHT, the objects can be looked up and retrieved. Thus complex data structures can be stored and the diverse applications of a social network are supported.

[1,2]Authors supported by the German Research Foundation, Research Group 733, "QuaP2P: Improvement of the Quality of Peer-to-Peer Systems" [1].

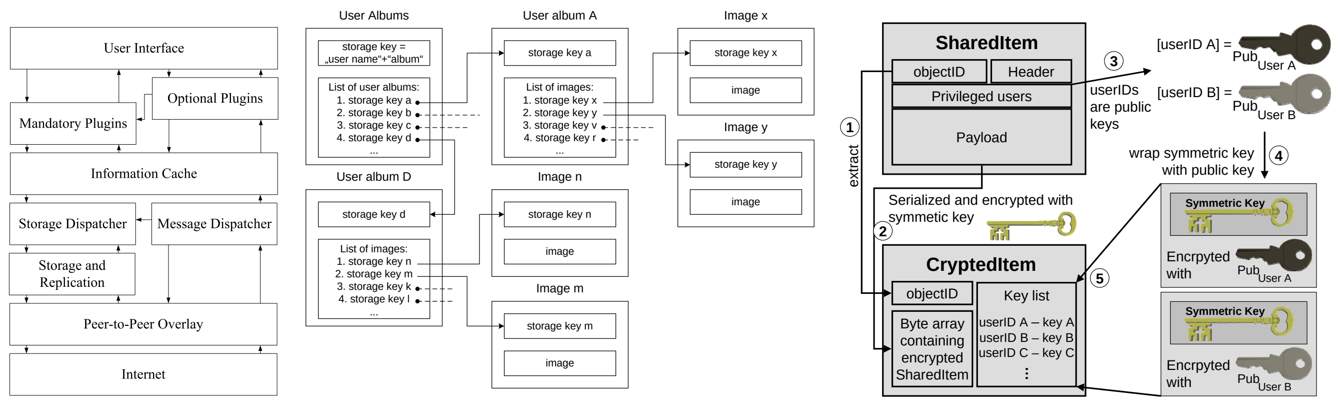
A. Security Requirements

After having briefly described our decentralized social network platform [2], we now focus on the security requirements.

1) Registration and Login: A registration phase is needed to grant new users access to the network and to create credentials for the user for later authentication. Users should be able to log on at every peer in the network, thus login credentials should be purely based on the knowledge of the user. After the registration, the user should be equipped with a valid and unique userID and authentication information. The authentication information should be stored confidential, available and with integrity.

The login functionality enables the (pre-registered) user to announce his status in the network. During the login process the user authenticates himself against the authentication information from the registration phase. As a result the joining of the node is announced in the network, and the node / pseudonym can further on be contacted by other nodes.

2) Access Control: We distinguish between user and group based access control, in both cases the security goals are similar. For all documents stored in the network the author should be able to mark *privileged* users, which are authorized to read these documents. Access to selected information of user specific information (e.g. profile details) or whole documents (e.g. photos) should be controllable. To manage groups with thousands of users a group based access control is needed. We call all storable data *SharedItems*.Access rights are dynamic and must be changeable at any time if the author of the document decides to do so. Access control aims to ensure the integrity, confidentiality and availability of SharedItems inside the community. SharedItems of users or groups must be available with expected service up time (e.g. 99,9 percent), thus the security solution must be compatible to common



(a) Plugin-based Architecture (b) Example of a Distributed Linked List (c) SharedItem and CryptedItem with Key List

Fig. 1. Data-centric Security for P2P-based distributed Data Structures

replication mechanisms and caching mechanisms. There must be no restrictions on the peers that store the data.

3) Secure Communication: During a live chat, all messages are directly sent to the users they are addressed to. For this communication, the sender and receiver must be authenticated, the communication itself must provide confidentiality and integrity. This wide set of requirements is challenging to solve in p2p systems, due to the peers' unreliability and autonomy.

II. A Security Framework for P2P-based Platforms for Social Networks

In this section we describe the design of our security framework for p2p based social networks. To summarize the idea, each user creates with his username and passphrase an asymmetric key pair. The public key is used as nodeID and userID in the network. Any communication is encrypted with the public key of the receiver, thus secure and authenticated communication can be provided once the nodeID of the receiver is known. For data storage and access control, we use a hybrid approach. All sensitive data is encrypted with a unique symmetric key, this symmetric key is encrypted with the public keys of the privileged users. The encrypted and signed data and the encrypted keys are stored as a package (SecuredItem) in the p2p network. Any node may retrieve and replicate this data, but only privileged users can decrypt it.

A. Registration and Login

In the registration process credentials for new users are created in a fully decentralized way. First, the user picks a (unique) user name and passphrase, which is used to generate an asymmetric key pair $Priv_A$, Pub_A. The numeric representation of Pub_A is used as nodeID and userID.

To join the network a request containing the node's information is send to a bootstrap node. A bootstrap node can be any formerly known node. It looks up the generated nodeID inside the network to prevent any nodeID collisions. If the object exists the user is already registered, thus next registration steps are skipped. If the object does not exist, the new user creates a minimal public profile, signs it and stores it in the p2p network. Through the signature, the profile is integer. The user is now equipped with a valid userID that will be the basis for later authentication and encryption processes inside the community since the userID is also his public key. Documents or data signed with the user's private key $Priv_A$ can now be validated.

For the Login process, user A recreates his key pair by entering his user name and his passphrase within the application. His userID is then derived from the just generated public key. The application sends a login request with the user's userID respectively public key to an available bootstrap node. The bootstrap node answers with information about further nodes to contact. This answer is encrypted, using the public key of the joining peer. The information is crucial to join, the joining peer must decrypt the data, thus to authenticate itself.

The presence of a user is depicted by a LoginItem, that is stored in the network. This signed object contains user's nodeID and his IP address. Every time a user logs in, he updates his IP address in the object. The signed LoginItem can be retrieved and verified by any other user. The nodeID/userID is further used to encrypt communication to this node (as it is a Public Key). Only the receiving node can decrypt messages that are encrypted in such a way. The concept of using the userID as a public key allows to established a simple PKI without any servers or certificate authorities.

B. Access Control

A user can read a SharedItem, create a new SharedItem or to alter an existing one. In each case he must prove his access rights to do so. We decided to use an Access Control Lists (ACL) based approach instead of Capability Lists, as ACLs can be sticked to data objects and allow an object-specific fine grained control and replication strategies. Each SharedItem that needs access control is encrypted with a object-specific symmetric key. To the SharedItem a data structure (*key list*) is added which holds copies of the encryption key of the SharedItem, wrapped (encrypted) with the public keys of the users who are allowed to access the item. The Shared Item in addition with the key list is signed by the author and named CryptedItem. CryptedItems contain all information to enforce access control, they can be replicated and cached. An overview on the SharedItem and CryptedItem is given in Figure 1(c). Next, we describe the access patterns.

1) Write and Read Access: To store a new SharedItem it is created with a timestamp and signed by the user for later verification of the author. Next, the user defines which other users should be allowed to read this item. If the user decides that only a set of privileged users should be allowed to read the item, he creates a symmetric key and encrypts the SharedItem, including its signature, with this key. See therefore steps 2 and 3 in Figure 1(c). This symmetric key is then encrypted with the public keys of the privileged users, which leads to n encrypted copies of the symmetric key for n privileged users (step 4). The encrypted copies of the key are then attached to the SharedItem (step 5), which is then signed and finally stored as a CryptedItem in the network. Each SharedItem (and thus also CryptedItem) has an objectID, which indicates where in the DHT the object will be stored (step 1). Using this ID, the object can be retrieved as well.

To alter an already existing object it has to be retrieved, modified and stored again. An ObjectID is created as a hash of the userID and some unchanging properties depending on the type of the SharedItem (e.g. hash(username + albumname)) (see [2]). As the ObjectID contains the username as well, any node can check whether a CryptedItem is valid or not using the ObjectID and the signature. For changing the privileged users of a SharedItem, only the attached keys have to be altered.

Any node can retrieve a CryptedItem from the p2p network. CryptedItems can be replicated and cached using any mechanism. However, only nodes listed in the key list can decrypt the SharedItem. If the retrieving node's ID is in the key list, the symmetric key is unwrapped and the item is decrypted.

2) Access Control in Groups: Inside a group, access to documents can be granted for all group members. This allows to use just one symmetric key for all accessible data inside a group. This symmetric key is created by the group founder at the time he establishes the group. At first, the founder creates a key list for his group where he stores the symmetric key encrypted with the public key of the group members. For each new member that joins the group, the administrator just adds a copy of the symmetric key, encrypted with the pursuant public key. This list is stored in the network, signed with the administrators private key to inhibit unauthorized write access. The objectID of this list is a hash of the administrators public key and the name of the group. A user can now store new SharedItems just as described above with the only difference that if he wants to make the item only accessible to group members, he encrypts it with the symmetric key of the group. For read access, a user accesses the key list of the group instead of the key list of a particular item. The protocol for read access is aside from that the same as described above.

C. Live Chat and Messaging

The live messaging functionality benefits from the design of making the public key of a user also his userID inside the network. User A wants to establish a secure connection to another user B for the purpose of a direct plugin to plugin communication, e.g. a live chat session. We use a hybrid approach for secure communication. User A creates a symmetric session key to encrypt his chat message to user B. User A sends the encrypted message and the symmetric key to user B encrypted with the public key of user B and signed with his own private key. With the signature both the integrity of the message can be checked and the sender verified. User B now verifies that the message is really from the sender with the given userID by verifying the signature of the message. If user B wishes to answer, he creates a secret key for the communication himself and wraps it with the public key of user A. Both users have a secret key for communication now. Each message sent between the users is signed by the sender and verified by the receiver.

III. Related Work

Freenet [4] is a p2p based website platform, providing anonymous and resilient website and data storage. Although anonymity is achieved, data access as security goal and user interaction as function is not in scope of Freenet. Skype [5] allows user-to-user voice over IP communication, but does not support decentralized data storage. In Skype a server is used for registration and management of the keys and buddy information. Zattoo [6] and PPLive [7] provide p2p based streaming but do not support direct user interaction.

In [8] the registration process is managed by super peers, which are more powerful than casual peers. They are responsible for bootstrapping new peers, but no further security goals. PAST [9] as storage module of FreePastry [10] focuses on data availability, but also does not contain an access control enforcement mechanism. OceanStore [11] and Cryptree [12] provide secure data storage and access, Plutus [13] extends this solutions by group access control. They focus strongly on traditional tree based file system structures, a scheme that does not apply in social networks with various complex interconnected data.

IV. Testbed Evaluation

We implemented the security framework for the p2p platform for social networks, which we presented in [2]. The prototype [14] implements the described solution. We used as p2p overlay FreePastry [10] and as mechanism for asymmetric cryptographic keys we use RSA [15] with a key length of 1024 bits. To comprise the modulus and exponent, we enlarged the ID space of FreePastry to 1088 bit identifiers. For the symmetric keys we use AES [16] with 128 bits. A signature is 128 bits in size as well. The described key sizes represent a configuration, that provides a desired security level for reasonable costs. All values are averaged over 100 runs on an Intel Core 2 Quad machine with 2.4 GHz and 3GB RAM.

A. Data Overhead

Table I shows the data overhead on basic messages. We started with an empty message, containing no text but only the header, storage key, receiver ID and an empty payload. Then we increased the message size by adding larger message text. We have a nearly constant absolute overhead, smaller than 2 KB coming from the duplication of the receiver information and the storage key and from the size of the empty CryptedMessage. Encrypting a basic message and turning it into a byte array does not increase its size perceptibly. The overhead of 2 KB will not affect the traffic speed or the storage space noticeably. Our approach is therefore an acceptable solution regarding the data overhead.

Table II depicts the data overhead on SharedItems. The size of an item does not affect the data overhead, therefore we varied the number of privileged users as parameter. The overhead grows with the number of privileged users as for each privileged user, a copy of the secret key is added to the CryptedItem alongside the users' userID. Each additional privileged user causes a data overhead of about 413 bytes. Still the relative overhead is acceptable even for 200 privileged users. The SharedItem we used, is a PhotoItem which has a standard size of 346 KB. However, any other item of arbitrary size would have the same absolute overhead.

The overhead we must deal with in this case is larger than the message overhead if we have more than one privileged user. However, 200 privileged users for a single object is a turning point of whether individual user-based access control should be replaced by group based access control. To keep the scenario of a social network in mind, in cases with 200 or more friends, it is recommendable to introduce group-based access. The management of group keys is similar to the management of individual user keys in the CryptedItem, same costs apply.

TABLE I

MESSAGE ENCRYPTION DATA AND TIME OVERHEAD

Msg. Size (bytes)	**Encryp. Msg. Size** (bytes)	**Overhead abs.** (bytes)	**Overhead rel.** (%)	**En-/Decryp. Time** (ms)
895	2794	1899	212,18	10 / 9
995	2906	1911	192,06	10 / 8
1395	3306	1911	136,99	11 / 9
1895	3802	1907	100,63	12 / 9
2895	4794	1899	65,60	14 / 10
3895	5802	1907	48,69	13 / 9
5895	7802	1907	32,35	12 / 8
10895	12794	1899	17,43	11 / 9

TABLE II

SHAREDITEM ENCRYPTION DATA OVERHEAD

Privil. Users	**Item Size** (bytes)	**Encryp. Item Size** (bytes)	**Overhead abs.** (bytes)	**Overhead rel.** (%)	**En- / Decryption** (ms)	**Key Wrap. ping** (ms)
1	346697	348159	1462	0,42	15 / 20	1
10	346715	351892	5177	1,49	25 / 21	4
50	346819	368524	21705	6,26	34 / 20	19
100	346969	389318	42349	12,21	54 / 19	37
200	347269	430922	83653	24,09	89 / 20	73

B. Time Overhead

We present the encryption and decryption times as an important metric for the costs of a practical security framework in Table I. The encryption and decryption time is around 12 ms and almost independent of the message size. Most to of the time is needed for administrative processes like obtaining the encryption keys and building the CryptedMessage.

For the time evaluation of the SharedItem, we can see that the encryption time rises linear with the number of privileged users. That is because the wrapping of the secret key with the public key of each privileged user takes about 0.36 ms time. Not surprisingly the decryption time is constant, as only one key has to be unwrapped in order to decrypt the item with the resulting symmetric key.

Data encryption is distinctly slower than message encryption when we must deal with many privileged users. Still, 89 milliseconds seem applicable for the encryption of items for 200 privileged users. Please note, that all used public keys were present in a *buddy keys* list, they were not needed to be retrieved from the network. That applies for the message encryption as well as for the item encryption we will investigate below. However, this is a reasonable step, as user knowing the privileged user(ID) also know the corresponding public key.

V. CONCLUSION

Social networks are very popular in these days, however client/server based solutions are expensive and do not scale. P2P-based platforms face several challenges, among the security requirements which we addressed in this paper. Our security framework for p2p-based social networks includes the support of user registration and a login process which allows further authentication of the users. Any user and application communication is confidential, integer and authenticated. We also presented a access control solution both for user-based access control and group-based access control. The security framework solves the security issues appearing in social networks. We implemented the security framework in our p2p-based platform for social networks, demonstrated its applicability and evaluated both its performance and costs. Evaluation shows that all security requirements were solved and the overhead in terms of space and time are low and reasonable in a p2p-based scenario.

REFERENCES

[1] DFG Research Group 733, "QuaP2P: Improvement of the Quality of Peer-to-Peer Systems," http://www.quap2p.de.

[2] K. Graffi *et al.*, "A Distributed Platform for Multimedia Communities." in *IEEE International Symposium on Multimedia (ISM '08)*, 2008.

[3] F. Dabek *et al.*, "Towards a Common API for Structured Peer-to-Peer Overlays," in *Proc. of IPTPS '03*, 2003.

[4] Freenet, "Freenet homepage," http://freenetproject.org/cgi-bin/twiki/view/Main/WebHome, 2001,.

[5] Skype, http://www.skype.com, 2004.

[6] Zattoo - TV to Go, http://www.zatoo.com/, 2007.

[7] PPLive - The Largest World Wide Internet TV Network, http://www.pplive.com/.

[8] W. Nejdl *et al.*, "Super-Peer-Based Routing and Clustering Strategies for RDF-Based P2P Networks," in *Proc. of WWW*, 2003.

[9] P. Druschel, "PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility," in *In HotOS VIII*, 2001, pp. 75–80.

[10] Freepastry, http://www.freepastry.org/FreePastry/.

[11] J. Kubiatowicz *et al.*, "Oceanstore: an architecture for global-scale persistent storage," in *Proceedings of the 9th International Conference on Architectural support for Programming Languages and Operating Systems*. ACM Press,, 2000, pp. 190–201.

[12] D. Grolimund *et al.*, "Cryptree: A Folder Tree Structure for Cryptographic File Systems," in *Proc. of SRDS '06*, 2006.

[13] M. Kallahalla *et al.*, "Plutus – Scalable Secure File Sharing on Untrusted Storage," in *In Proc. of USENIX FAST*, Mar. 2003.

[14] LifeSocial.KOM, http://www.lifesocial.org/, 2009.

[15] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[16] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - the Advanced Encryption Standard*. Springer, 2002.