

Practical Security in P2P-based Social Networks

Kalman Graffi, Patrick Mukherjee, Burkhard Menges, Daniel Hartung, Aleksandra Kovacevic, and Ralf Steinmetz
Multimedia Communications Lab¹ and Real-Time Systems Lab², Technische Universität Darmstadt, Germany
The Norwegian Information Security Laboratory, Gjøvik University College, Norway
Email: {graffi¹,sandra¹,steinmetz}@kom.tu-darmstadt.de, mukherjee²@es.tu-darmstadt.de, daniel.hartung@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 for 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

Peer-to-peer (p2p) networks evolve as alternative to client/server based architecture in a wide area of applications. The technology promises quality comparable with client/server solutions but far lower administration costs, as the load is shared among the network participants. However, security aspects of applications are harder to solve following the p2p paradigm than in client/server based architecture.

As a next application area for p2p technology the field of social networks has been identified. Social networking sites are web-based platforms allowing the users to publish personal profiles, link themselves, post pictures and blog entries, join groups and search for friends. Several hundred millions of users participate in today's social networks like Facebook [2] or MySpace [3]. However, due to centralized character of the platforms, high server maintenance cost may lead to the fall of popular social network sites. A p2p-based approach solves the load and cost issues but leads to new challenging security issues for secure communication and data storage in decentralized platform.

In Subsection I-A we recapitulate our solution for a p2p-based platform for social networks which can be reviewed in detail in [4]. We discuss the distributed data structures and derive in Section II the requirements and goals for a security framework for p2p-based platforms for social networks. We address this requirements through a discussion of related work and applications in Section III. In Section IV we present in detail our solution for authenticated network participation, secured data storage with fine grained data access control and authenticated, integer and confidential communication. An evaluation of the proposed solution is given in Section V. We show that the proposed solution solves the requirements for

a secure p2p-based platform for social networks and comes with low traffic and computation costs.

A. A P2P-based Platform for Social Networks

In this subsection we summarize briefly our approach for a p2p-based platform for social networks, more details can be found in [4]. Social networks provide the 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 to communicate with friends and groups through a messaging system. We split this wide set of functionality into individual modular functionality blocks, like friends management, photo management and proposed a plugin based architecture, as shown in Figure 1(a). Plugins are able to communicate with each other, so that more advanced plugins can be created reusing the functionality of other plugins. In order to do so, plugins can communicate with each other using a Message Dispatcher, dispatching messages between the plugins. Plugins operate also on an Information Cache which manages data storage and retrieval and also quickens the access to previously requested data as depicted in Figure 1(b). Data storage is completely decentralized through the usage of a structured p2p overlay and a corresponding Storage and Replication layer. The structured p2p overlay provides the functionality of ID-based routing, compliant to the Key-Based Routing (KBR) interface proposed by Dabek et al. in [5]. The Storage and Replication layer provides the functionality of replicated data storage and local storage management. This layer also implements thus a distributed hash table (DHT). 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 proposed in [4] a distributed data structured based on lists, as seen in Figure 1(c). 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 the 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.

II. SECURITY REQUIREMENTS

After having briefly described the architecture of our decentralized social network platform [4] and the data structures used, we now focus on the security requirements in such an environment.

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

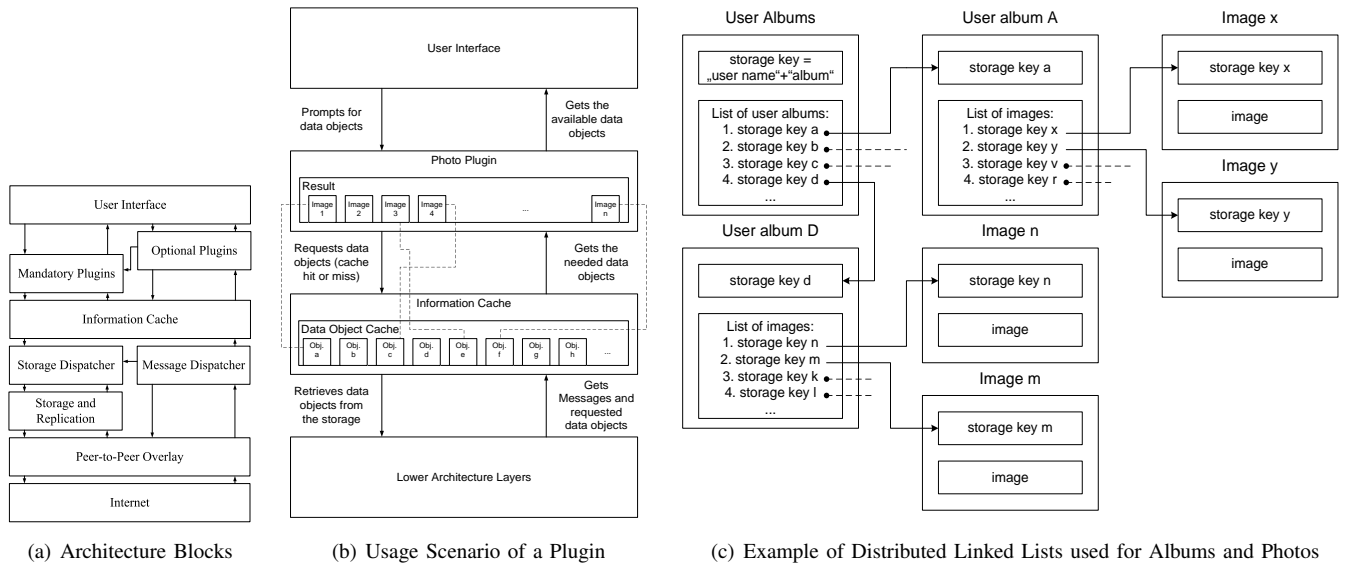


Fig. 1. Layer-based Architecture for Multimedia Online Communities

A. Registration

A registration phase is needed to grant new users access to the network and to create credentials for the user for later authentication. Both open and closed (access controlled) networks should be supported. Before the registration, the user starts with nothing but his name or a pseudonym he wants to use as a user name inside the community. After the registration, the user should be equipped with a valid and unique userID and authentication information with which only he is able to log on to the community network. The authentication information should be stored confidential, available and with integrity. Users should be able to log on at every peer in the network, we do not assume that users are willing to transport sensible data. We do not assume that users want to prove their identity to a Certificate Authority (CA), users may decide to use a pseudonym instead. In the p2p scenario we assume that peers join the network and fail, still the registration service must be available all the time. The result of the registration phase is a credential, proving authenticity of the user, allowing him to use the network for a given time and to authenticate against other peers.

B. Login

The login functionality enables the user to announce its status in the network. This steps requires that the user registered before at the social 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. Only the user to whom the authentication information belongs must be able to use it to authenticate against the network. The authentication information in the network must be stored integer, confidential and with high availability.

C. Access Control

Access Control surely takes in the biggest part of the security considerations. We must distinguish between situations where only one user and a group of users have access to a file or a set of data. Nevertheless the security goals we must consider are similar in each case.

Examples for access control:

- **Profile:** Some information of a users profile should only be accessible to a dedicated set of users. A user might for example only allow his friends to see private data like his email address, his phone number or his instant messenger name.
- **Messaging:** When sending a message to a user only the receiver and the sender should be able to read this message.
- **Photos:** Every user can store photos in the network and share them with other users. As he can decide which users or groups of users are allowed to view them, it must be guaranteed that only the people he has authorized can do so, these are called *privileged users*.
- **Groups/Networks:** Inside a group the same issues appear as mentioned above. Here again one can send messages or store and share files. Each group has a main page where a brief description of the group can be given and messages of group members can be posted. However, some of this content may only be accessible for members while other content may be public.

We call all storable data *SharedItems*, they are stored in the p2p network. We address files, photos and profile data as *SharedItem* during the security goal analysis. Since all *SharedItems* are distributed over the whole network, locally stored files also need an access control concept.

In some areas, users have the possibility to decide who has the right to view their shared data. These access rights

are dynamic and must be changeable at any time if the user decides to do so. These privileged users must be stored for each area where one can define them and only the author must be able to access them.

The delete function is of special importance because besides the common problem to prove ones right to change data, it has to be guaranteed that data is no longer available inside the community if the author or another privileged user deleted it.

Access control aims to ensure the integrity, confidentiality and availability of SharedItems inside the community. The author of a SharedItem defines which others users are allowed to alter it. In most cases this is only the author himself. It must be warranted, that no other user can alter SharedItems if the author has not given him the authority to do that. All data must be stored confidential, the author must be able to define users or groups of users which should be able to read specific SharedItems. Any unauthorized access must be inhibited. The data structure defining the privileged users with their access rights must itself be protected against unauthorized access. SharedItems of users or groups must be available at 99,9 percent of the time. A security solution must be compliant to common replication mechanisms and caching mechanisms. There must be no restrictions on the peers that store the data, thus SharedItems may be stored on unreliable and even malicious peers. Still confidentiality and integrity must be guaranteed.

D. Secure Communication

During a live chat, all messages are directly sent to the users they are addressed to. Though no storage inside the network is used, it must be assured that no third person can access or alter the message while it is sent. Before starting the conversation the users must authenticate each other.

III. RELATED WORK

Freenet [6] is p2p based website platform, providing anonymous and resilient website and data storage. However, user interaction is not supported and covered by the integrated security framework. Skype [7] allows user-to-user voice over IP communication, but does not support decentralized data storage. In file sharing typically no specific security requirements are stated. Zattoo [8] and PPLive [9] are p2p based streaming applications which do not support direct user interaction.

In [4] we presented a p2p-based architecture for online community platforms. It supports the same functionality as client/server based solutions, but is completely decentralized. In this paper we describe in detail how

IV. 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

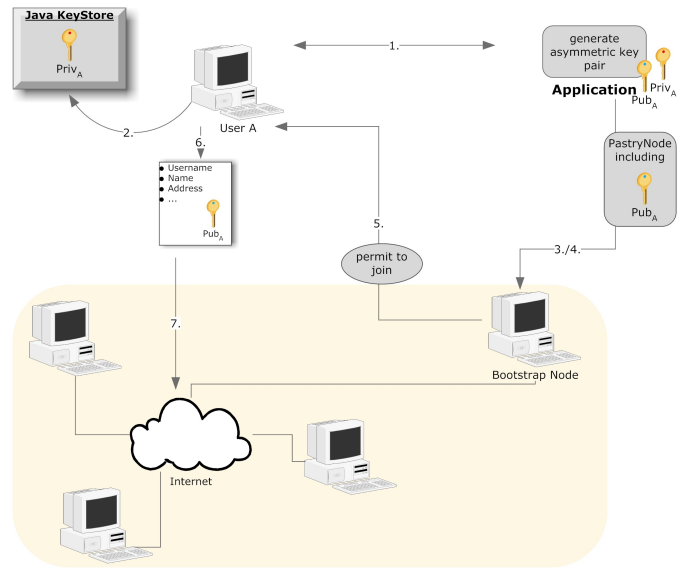


Fig. 2. Registration process

communication can be granted 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 access and replicate this data, but only privileged users can decrypt the data.

In the following we describe the solution in detail and present an evaluation in Section V-B.

A. Registration

Goal of the registration process is to create credentials for new users, that can be used later on to authenticate the user. Both an open and closed network should be supported. For the open network, we cannot assume a centralized credential provider, thus we developed a fully decentralized solution, which can be adopted to closed networks as well.

In order to register at the social network, a new user must pick an user name and passphrase, this is the root of trust. The user name and passphrase is used to generate an asymmetric key pair $Priv_A, Pub_A$. Optionally, the private key $Priv_A$ can be securely stored on the local computer of user A in order to save time at following login steps. However, for security reasons this step may be omitted. The numeric representation of Pub_A is used as nodeID and as userID.

The application now contacts a bootstrap node with the request to join the network. It sends the generated node information alongside the request. Bootstrap nodes may be the addresses of nodes that were previously online. The bootstrap node 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 presence of the user is depicted by a LoginItem, that is stored in the network. This signed object contains user's nodeID, its IP address. Every time a user logs in, it edits this object and updates the IP address in the object.

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. The possession of a specific key pair and hence the belonging to a certain pseudonym can easily be proved by simply challenging the specific user with a message encrypted with his public key Pub_A . It is now also possible to validate every document or data signed with the users private key $Priv_A$. This serves the purposes of authenticate oneself as the author respectively owner of data.

B. Login

The login process is quite similar to the registration process. The main principle of proving ones identity is the verification of the possession of the private key associated with ones userID which is at the same time ones public key. Again there is no need of a central institution as the login process is designed completely decentralized. Following process leads to the login of the user A. A user is logged in, once the LoginItem, which is stored in the p2p network, contains its valid IP address and the joining node has sufficient contacts in the p2p network.

User A loads his asymmetric key pair by entering his chosen username and passphrase if the key pair is stored on his computer. If he wants to use the login functionality on a foreign computer or his credentials were not save, he 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 easily, as it is a numerical representation of the public key. The application contacts an available bootstrap node inside the existing p2p based social network. There are some predefined bootstrap addresses and the cached addresses of previously online nodes.

The application sends a login request alongside its userID respectively public key. The bootstrap node answers with information about further nodes to contact. This answer is encrypted, using the public key of the joining peer. As the information is crucial to join, the joining peer must be able to decrypt the data, thus to authenticate itself.

The user is now a valid node inside the p2p network. As a next step, the new node retrieves its LoginItem, updates the IP address in this data, signs it and stores it again in the network. Any other user, can now lookup the contact information of the joined node using the DHT-functionality of the structured p2p network. The signed LoginItem can be retrieved and verified. The IP address tells how the user corresponding to the userID / nodeID can be contacted. The nodeID/userID is further used to encrypt communication to this node. Only the receiving node can decrypt messages that are encrypted in such a way.

The concept of using the userID as a public key or vice versa, allows every user inside the community to obtain every

other users public key, as it just requires to look up the users Id. So we have established a simple PKI without any servers or certificate authorities. That leads to the issue of trust inside the community. However, each person is allowed to join the community and can choose his user name as he likes. The truth about the real identity of a user can therefore only be determined by the users friends, who know his pseudonym or can make sure the user is who he claims to be by talking to him.

C. Access Control

Mature access control in a distributed network of unreliable nodes is a challenging task. All accessible data in the p2p network we call SharedItem. A user can now either desire to just read a SharedItem, to create a new SharedItem or to alter an existing one. In each case he must prove his access rights to do so. We can differentiate between two general concepts for access control of data, Access Control Lists (ACL) and Capability Lists. We decided to use an ACL based approach, as ACLs can be sticked to data objects and allow an object-specific fine grained control. Moreover assigned access rights for SharedItems are efficiently determinable and the revocation of these rights can also be efficiently implemented.

Our ACL does not really control the access to an item, but nevertheless determines if a user can read (encrypt) an item he downloaded from the network. Each SharedItem that needs access control is encrypted with a object-specific symmetric key. A data structure added to the SharedItem created 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. As this concept is not the classic ACL, we will address it with *key list* in the following. 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 3.

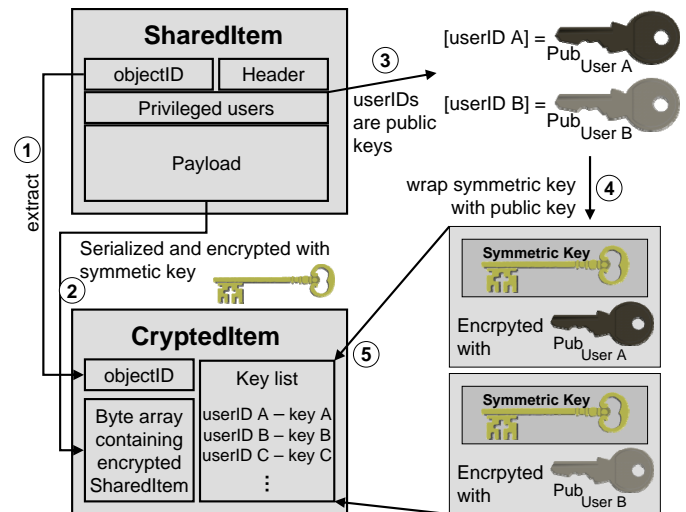


Fig. 3. CryptedItem with Key List

In the following we describe the various access patterns.

1) *Write Access:* In order to create a store a new SharedItem in the network first the p2p social network application creates a SharedItem. For later unique identification and version control a timestamp is added to the item and the user finally signs the item with his private key so he can be verified as the author using his userID as public key. The signature is later stored in the header. Now the user defines which other users should be allowed to read this item. Figure 3 depicts the encryption process of a SharedItem. If the user wants to allow everybody in the community to read the item, no further access control is needed and the item can be stored in the network. If he decides that only a group of other users, e.g. some of his friends taken from his friends list, should be allowed to read the item, he creates a symmetric key within the application and encrypts the SharedItem, including its signature, with this key. See therefore steps 2 and 3 in Figure 3. This symmetric key is then encrypted with the public keys of the users who are allowed to read the item so that in the end there exist n encrypted copies of the symmetric key for n privileged users (step 4). The encrypted copies 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.

a) *Modifying existing objects:* The alteration of an already existing object is done by retrieving it, modifying it and then storing it again. In order to prohibit malicious users to override data that do not belong to them, the userID is involved in the ObjectID itself. The 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)). 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.

b) *Revocation and adding of new privileged users:* For changing the privileged users of a SharedItem, only the attached keys have to be altered. Therefore the corresponding CryptedItem is loaded, the key list updated, the new CryptedItem is signed and stored again. Please note, that a malicious node may also try to store a modified packet. However, integrity of the information is prevailed as the necessary signature cannot be forged and the modified data is obviously invalid.

2) *Read Access:* Now we will inspect the security measures used when users try to read a SharedItem of another user or of their own. Please note, that any node can retrieve any CryptedItem from the p2p network using the DHT. Thus, CryptedItems can be replicated and cached using any mechanism. However, only nodes listed in the key list can encrypt the SharedItem.

Following process takes place to read the SharedItem. The user retrieves the desired CryptedItem using the DHT. The application downloads the item and checks if the users public key appears in the current key list of it. If the key is in the

list, the symmetric key is unwrapped and the item is decrypted. The item is then cached locally and can be stored for further use if the user wants to.

3) *Access Control in Groups:* The access control inside a group is just slightly different from the user-based access control. As the options for determining the privileged users are smaller inside a group, the item encryption can be handled more simple.

Inside a group, the only possibilities when publishing a SharedItem, are either to make it publicly available or to make it only available for all group members. This circumstances allow it to use just one symmetric key for all accessible data inside a group. This symmetric key is created by the group administrator at the time he establishes it.

To become a member of a group, one has to send a request to the group administrator. When the administrator lets the first user join his group, he creates an key list for his group where he stores the symmetric key encrypted with the public key of the new user. For each new member that joins the group, the administrator just adds a copy of the symmetric key, encrypted with the particular 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 can be created as 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. The item is still signed with the users private key to ensure write access control. 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.

D. Live Chat and Messaging

The live chat and messaging functionality also benefits from the design of making the public key of a user also his userID inside the network. We use this fact to implement a key usage protocol and more important to authenticate the desired communication partner.

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 within the application for the purpose of secure communication. He then encrypts his chat message he wants to send to user B with this symmetric key. 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. If user B is currently online, the message will be delivered directly and if B is offline, the message will be stored in the network. 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

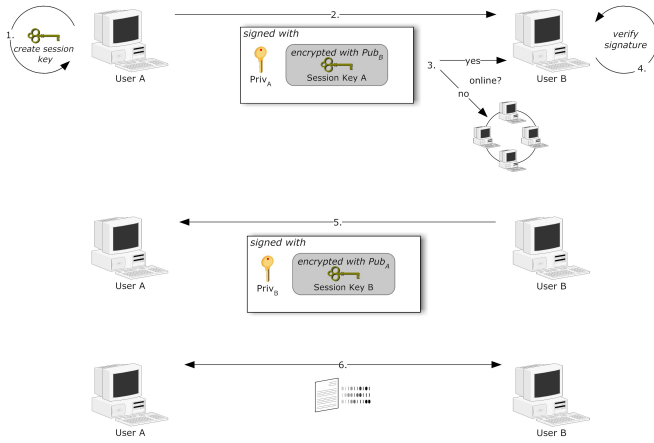


Fig. 4. Secure Communication

and wraps it with the public key of user A. Both users have a secret key for communication now. Each sent message between the users is signed from the sender and verified from the receiver.

For the message encryption process we need besides the storage key also to extract the userID of the receiver and add it to the CryptedItems. The key list, containing the encrypted secret keys for decryption has only two entries in a CryptedItems. It holds the encrypted key for the receiver and one copy for the sender himself, as he may wish to read his sent message some time at a foreign computer.

BasicMessages are not only sent from user to user but also from plugin to plugin. Such messages can contain requests or information for maintenance or status information. All these messages are encrypted and as some plugins in the application access received messages twice, a copy of the encryption key encrypted with the senders public key inside the key list is needed for such purposes as well. Thus all messages can be decrypted by authorized users all the time.

In this Section we presented our solution for a security framework for a p2p-based social network platform. In the next section, we analyze the security features of the solution and discuss corresponding costs.

V. EVALUATION

We implemented the security framework for the p2p platform for social networks, which we presented in [4]. The prototype¹ implements the described solution. We used as p2p overlay FreePastry [10], which supports the desired DHT functionality. As mechanism for asymmetric cryptographic keys we use RSA [11] with a key length of 1024 bits. The public key consists of two components: the modulus and the exponent. Both are needed to generate a key, that can be used to encrypt data and verify signatures. To hold both values, we enlarge the ID space of FreePastry to 1088 bit. For the symmetric keys we use AES [12] with 128 bits. A signature is 128 bits in size as well. Please note that using elliptic curve

based cryptography, would allow us to use smaller keys yet providing the same security level. The described key sizes represent a configuration, that provides a desired security level for reasonable costs. However, the used mechanisms and key sizes can be adopted to the needs of the application scenario.

A. Security Analysis

In this subsection we discuss and show, how the described solution fulfills the security requirements stated in Section II.

1) Analysis of Registration Requirements:

a) *Available registration functionality:* The registration of a new user only requires the presence of an available member of the p2p network to act as a bootstrap node. It can be assumed, that the probability of all existing members being down is very low. Therefore the availability of the registration functionality is given with high probability.

b) *Secure communication during registration:* The communication between the bootstrap node and the new user in the consists of the registration information including its nodeID, which the user sends to the bootstrap node. The main information is the nodeID of the new node which is the public key of the user and its unique userID. All messages in the p2p network are signed and encrypted in the case that the receiver is known. This is the case during the registration process.

c) *Usage of pseudonyms:* The user can create his own desired user name after he generated a public key respectively a valid nodeID. This user name is stored in the users profile. The

d) *Confidential, available and with integrity stored authentication information:* An userID is the public key of the corresponding user, it has be derived from his username and a passphrase. This information is root of trust in the system and authenticates the user, thus this knowledge can only be retrieved by social attacks. The result, the public key, is publicly available.

e) *Unique userIDs:* The username and passphrase are directly linked to the userID. Random ID collisions are very unlikely in an ID space with 1088 bits, thus userIDs are unique.

f) *Registration results in an authentication credential:* The result of the registration is a credential (public key), proving authenticity of the user, allowing him to use the network for a given time and to authenticate against other peers. The equality of userID and public key achieves this goal. Please note, our solution support pseudonymous (userID-based) authentication not real personal identities.

g) *Secure storage of user data:* As the user has the possibility to encrypt his data, confidentiality is guaranteed. Integrity is assured by the signature the user creates for his profile data. The replication of stored data provided by the storage and replication component PAST [13] of FreePastry [10] ensures data availability.

2) Analysis of the Login Requirements:

a) *Available login functionality:* The login of a existing user only requires the presence of an available node in the p2p network. It can be assumed, that the probability of all existing members being down is very low.

¹For more information please see www.lifesocial.org

b) *Login with authentication information:* During the login process the new user provides his userID and receives a list of further nodes to contact. As all communication in which the receiver is known is encrypted, only joining peers with valid userIDs can decrypt the information. Peers which cannot decrypt messages cannot join and stay in the network, as they cannot comply to maintenance protocols.

c) *Full access to the network after successful login:* After a successful login the user will be able to access every data and functionality that are associated with his public key. Details we discussed in in Subsection IV-C.

3) *Analysis of the Access Control Requirements:*

a) *Data structure defining the privileged users with their access rights:* The key list (ACL) added to each item lists all users with access right.

b) *This ACL itself must be protected against unauthorized access:* Each CryptedItem is signed by the author, which makes an undetected change by anyone who does not possess the signing private key impossible. As the CryptedItem contains the key list, a change of the key list would result in a change of the CryptedItem, thus integrity is provided. The CryptedItem contains the encrypted SharedItem, which is stored confidentially.

c) *Authentication of users for access control:* Any user may retrieve the CryptedItem, but only for the users listed in the corresponding key list an encrypted key is stored in the CryptedItem. Thus only users, that were authorized are able to decrypt the SharedItem.

d) *Concept to ensure availability of data:* PAST [13], the storage and replication module of FreePastry [10] ensures data availability. However, any other replication mechanism can be used, as the security framework does not state requirements on the replication layer.

e) *Confidentiality and integrity for locally stored data:* CryptedItems can only be decrypted by nodes listed in the key list, thus confidentiality is provided. Any modification that is not signed by the author is detected, thus integrity is provided. A malicious node can only delete CryptedItems undetected, this effect is covered by the storage and replication module which assumes regular peer failure.

4) *Analysis of the Live Chat Requirements:* Please note, that the live chat represents all kinds of plugin to plugin communication.

a) *Authentication of communication partners:* Each message between the two communicating users is signed with the private key of the sender and encrypted with the public key of the receiver (which is its userID). As the keys are pre-known, man in the middle attacks cannot be done.

b) *Confidential communication with integrity:* We use hybrid cryptography to provide the security level of asymmetric cryptography with the low costs of symmetric cryptography. In the first message of a plugin to plugin communication a symmetric secret key is created and used to encrypt the messages. The key itself is encrypted with the receivers public key and added to the message. The message is signed with the private key of the sender. The message contains the

signature and the userID of the sender, thus the integrity of the message can be proved. Every following communication is now encrypted with the symmetric key, which lowers the data and time overhead in comparison to use asymmetric cryptography.

B. Testbed Evaluation

In this Section we will both analyze the security performance and the costs, how our security features affect the runtime of the application and the storage space in the network. We will see how much additional time is needed for encryption and decryption of SharedItems and BasicMessages and how much additional data overhead is produced because of the encryption of the two data types.

All measures were taken on a Intel Core 2 Quad machine with 2,4 GHz and 3GB RAM. All time values are average values of 100 runs.

1) *Data Overhead:* We will now take a look at the data overhead which are caused by our security features. First present the overhead for messages and then for storable objects. Table I shows the data overhead on BasicMessages. 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 can see, that a different message size does not affect the absolute data overhead. We have a nearly constant 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 BasicMessage 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 500 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, the same costs apply.

Message Size (bytes)	Encryp. Message Size (bytes)	Overhead abs. (bytes)	Overhead textbfrel. (%)	Encryption Time (ms)	Decryption Time (ms)
895 (empty message)	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 I
MESSAGE ENCRYPTION DATA AND TIME OVERHEAD

Privileged Users	Item Size (bytes)	Encryp. Item Size (bytes)	Overhead abs. (bytes)	Overhead rel. (%)
1	346697	348159	1462	0,42
10	346715	351892	5177	1,49
50	346819	368524	21705	6,26
100	346969	389318	42349	12,21
200	347269	430922	83653	24,09

TABLE II
SHAREDITEM ENCRYPTION DATA OVERHEAD

2) *Time Overhead*: We present the encryption and decryption times as an important metric for the costs of a practical security framework. We first present the times for the encryption of a BasicMessage. Table I shows also the results of the time measurements.

The results show, that the encryption and decryption time is independent from the message size. That can only mean that the most time is needed for administrative processes like obtaining the encryption keys and building the CryptedMessage.

An encryption time of around 12 milliseconds appears to be fast and does not affect the functionality or performance of the messaging plugin of our p2p based platform for social networks. Our message encryption approach is hence applicable regarding the time overhead.

Finally we present the results of the time evaluation of the item encryption, displayed in Table III. Again we used the same item size as in Section V-B.1.

Privileged Users	Encryption (ms)	Decryption (ms)	Key Wrapping (ms)
1	15	20	1
10	25	21	4
50	34	20	19
100	54	19	37
200	89	20	73

TABLE III
SHAREDITEM ENCRYPTION TIME

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 knows the corresponding public key.

VI. 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 the security requirements which we addresses 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] FaceBook, <http://www.facebook.com/>, 2007.
- [3] MySpace, <http://www.myspace.com/>, 2007.
- [4] K. Graffi *et al.*, "A Distributed Platform for Multimedia Communities." in *IEEE International Symposium on Multimedia (ISM '08)*. IEEE, 2008.
- [5] F. Dabek *et al.*, "Towards a Common API for Structured Peer-to-Peer Overlays," in *Proc. of IPTPS '03*, 2003.
- [6] Freenet, "Freenet homepage," <http://freenetproject.org/cgi-bin/twiki/view/Main/WebHome>, 2001,.
- [7] Skype, <http://www.skype.com>, 2004.
- [8] Zattoo - TV to Go, <http://www.zattoo.com/>, 2007.
- [9] PPLive - The Larges World Wide Internet TV Network, <http://www.pplive.com/>.
- [10] Freepastry, <http://www.freepastry.org/FreePastry/>.
- [11] 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.
- [12] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - the Advanced Encryption Standard*. Springer-Verlag, Berlin Germany/ Heidelberg, Germany / London, UK / etc., 2002.
- [13] P. Druschel, "PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility," in *In HotOS VIII*, 2001, pp. 75–80.