Digital Object Identifier: 10.1109/ICDCSW.2013.20

URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6679862

SoNet – Privacy and Replication in Federated Online Social Networks

Lorenz Schwittmann, Christopher Boelmann, Matthäus Wander, Torben Weis
University of Duisburg-Essen
Duisburg, Germany
wimi@vs.uni-due.de

***Abstract*—In this paper we propose a federated online social network (OSN) which focuses on user privacy and data availability. All user content is encrypted and decrypted on end-user devices, hiding the content from the OSN providers. The social graph is hidden from the OSN provider by employing a novel aliasing approach and using secure algorithms for mutual friendship establishment. Usernames are mapped to friend-specific aliases, which reduces the amount of information a provider can gather from analyzing these identifiers. Users authenticate to each other without revealing their identities to a potential attacker. The proposed system allows for user interactions between independent OSN providers. To improve data availability we use a replication scheme which does not jeopardize the obfuscation of the social graph. Our approach differs from existing works mainly by the social graph obfuscation in combination with replication.**

I. Introduction

During the last years the number of users of *Online Social Networks* (OSN) has sharply increased. These networks are incompatible with each other: if a user wants to get in contact with friends on different OSNs he has to register in each network separately. This circumstance may be a more important OSN selection criterion than e.g. the quality of service, potentially impeding diversity and innovation. As a consequence, this had lead to an accumulation of huge user bases on a few OSN providers.

OSN providers collect large amounts of personal data which they use for advertisement or other purposes. Altough the *terms of service* (TOS) agreement between the user and the OSN provider sets legal limitations for the usage of personal data, the user has no means to check for compliance with the TOS. From the user's point of view, privacy needs to be enforced by technical measures. With such technical measures the OSN provider serves as online data storage for encrypted private information but without access to the private keys. Besides the actual OSN content like messages or images we also consider the social graph as private information. The social graph reveals with whom a user interacts, possibly sharing economic or other interests.

In this paper we propose a federated OSN approach. Users may become their own data storage provider or choose a commercial one to rent storage capacity. This data storage is used to share encrypted OSN content. Storage servers exchange data via a network protocol, enabling user interactions which span different OSN providers.

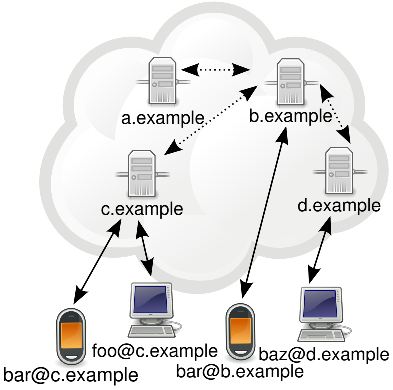

Figure 1. An example of a federated OSN

Based on the experience of Diaspora, we consider server availability as a key issue in federated OSNs: Bielenberg et al. discovered that 50% of the Diaspora servers have more than 50% downtime [1]. Although Diaspora is different by design than our approach, it uses also a federated server infrastructure. We are therefore using data replication between OSN providers to cope with server downtimes and to reduce query delays.

If a privacy-aware OSN is to succeed, it has to provide a similar level of comfort and usability as an existing centralized one. Even if employing cryptographic mechanisms on the end-user device, the system must be efficient enough to run on mobile phones. For further reference, our requirements are privacy (**R1**), availability (**R2**) and efficiency (**R3**).

II. Architecture

The basic architecture of our system consists of independent servers which communicate with each other using a federation protocol. Users choose one of these servers and connect to this host exclusively. If two users are to exchange a message, it has to pass one or two servers depending on whether both users chose the same provider or not.

This approach is similar to existing federated services like email or the Extensible Messaging and Presence Protocol (XMPP). Users are identified likewise by a *useridentifier@serverdomain* scheme. Figure 1 illustrates an example of such a system.

All data objects exchanged between users are end-to-end secured by cryptographic operations. Servers therefore are not able to interpret them but merely forward them as opaque binary blobs. Hence we separate the system into two layers: a federation protocol used between servers to exchange data and the actual OSN protocol used between clients. Since clients do not connect to each other, the OSN protocol is tunneled through the federation protocol. With this separation in place, the storage could be used for other collaborative applications in parallel with the OSN.

A. Storage Servers

Servers offer clients methods for storing and retrieving data objects. HTTPS is used to provide authenticity, privacy and integrity between client and server.

The interface provides access to a rudimentary filesystem in which data objects are stored as files in directories. Each data object is accessible using a unique path. Such a path always starts with a full user identifier and a local path.

The user identifier contains the server's address an object will be stored on. Clients can access data objects by querying their server. If the host part of the user identifier is not the same as the provider's address, the server will fetch the data object from a remote host and forward it to the client.

From a server's point of view, data objects consist of metadata (author, path, groups) and the actual data payload.

Clients can establish a permanent connection and subscribe for updates on a given path. After this, the server will push notifications about any changes in this path.

By default, only the user himself (in this context also called *owner*) may write to his storage. He can grant access to other clients by adding their user identifier and a symmetric secret to a list on his server. After this, the owner can define a set of writable paths for groups of friends. It is also possible to allow write-access to a certain path for all users. The server will enforce this policy by checking author/location combinations of incoming data objects.

The servers of different providers use a federation protocol. Like the client protocol it uses HTTPS. In contrast to client-server communication, the requesting host uses an X.509 server certificate to authenticate in TLS. Thereby authenticity between servers is ensured.

Servers exchange data objects written by their users. Servers are *authoritative* for objects whose path starts with one of their user identifier. If a user writes an object on a server it is not authoritative for, the server will *push* that object to its destination server.

If a server is authoritative for a user-written object, it pushes a notification about this new object to all servers hosting a user in the author's friend list. These remote servers decide whether they will pull that object or wait until one of its client actually requests it. If one of its clients has subscribed for the object's path, it will always pull the object. Otherwise heuristics are used and only small files below a certain threshold are pulled immediately. That way servers act as caches for objects of their user's friends.

B. OSN Features

Based on this federated architecture we have developed a distributed OSN. In the following section, we will list some of its features and how they are mapped to the underlying federation system. Other features which are mappable in a likewise manner have been omitted due to space constraints.

1) Circles: We have adopted the idea of circles from Google Plus. Each circle is a group of friends which has access to different objects of the storage. If a data object is encrypted, a member of a circle can only decrypt it if that circle is mentioned in the object's *groups* attribute. This allows fine-grained access control.

2) Posts: Users can create posts to be shared in specified circles. Posts by $a@X$ are data objects stored in /$a@X$/_s/posts/ using a generated post-id. Posts can be commented by other users. Given a post-id p, the comments are stored in /$a@X$/_s/posts/p/comments/.

3) Chat: Besides sharing content in circles, there is also the possibility to exchange messages with only one friend. This is implemented in chats. Each chat message m is associated with a friend $b@Y$ and stored in /$a@X$/_s/chats/$b@Y$/m.

C. OSN Security

We use a hybrid cryptosystem to enforce confidentiality. All content produced by users (posts, chats etc.) is client-to-client encrypted. For each entry, a random *entry key* is generated to encrypt it.

There is a *circle key* for each circle. This key is known to the circle's members and can be used to share content with them. For this purpose, the *entry key* is encrypted using the *circle key* and stored in the header of that entry. If content is shared in several circles, the *entry key* is encrypted once for each *circle keys*. There is a *key identifier* for each encrypted circle key to tell them apart.

Given such an identifier I, client $a@X$ can retrieve a circle key of author $b@Y$ by fetching it from /$b@Y$/_s/keys/circle_I. In this container, the circle key is encrypted for each of $b@Y$'s friends in that circle using their public key. These identifiers are consecutive integers starting from 0 and collide therefore for different users.

The expensive asymmetric decryption of circle keys can be relativized by caching decrypted circle keys. This is possible, because they change seldom. The circles whose keys have been used to encrypt this entry are stored in the data object's groups attribute. This allows clients to query only for data objects which they can decrypt.

Comments are handled differently: If Alice comments a post by Bob it is undesirable for a friend of Bob who is not a friend of Alice to be able to read it since depending on the context it might reveal information about Alice. Only those who are both a friend of Alice and were able to read the post in the first place should be able to read that comment. Therefore comments are encrypted using a hashed concatenation of both entry key and one of Alice's circle keys.

As soon as a friend is removed from a circle it is crucial to prevent him from reading any further content shared in this circle. Since he already possesses the associated circle key, we have to generate a new one with a new key identifier and distribute it by writing a new encrypted circle key for each remaining friend in that circle.

Besides circle-shared entries there is also content which is shared with a single friend. Chat is an example for such an access model. In that case, the entry is not encrypted using the circle key but a friend-specific symmetric key. This key can be accessed by the recipient of the message $b@Y$ at /$a@X$/_s/keys/$b@Y$/f2f. As circle keys, this friend-to-friend key is encrypted with $b@Y$'s public key.

So far key distribution has been discussed for reading clients. However, users are accustomed to using different devices for their OSN activities. To participate in this OSN, a client has to be in the possession of all keys required for writing as well. Therefore clients store a copy of each key on the server. This copy is encrypted using a symmetrical user defined password. From a user's perspective, the application asks for this password at login as it is known from existing OSNs.

III. Social Graph Obfuscation

To hide all information that might hurt the user's privacy or unveil social interactions from other people we describe in this section how social graph obfuscation is achieved in our OSN architecture.

To obfuscate relationships and interactions between users within the OSN we hide bidirectional links by using *single direction aliases* (SDA) that are unique for each direction of user-to-user relationships instead of user identifiers. Aliases are random strings of sufficient length that are generated by servers to hide user identities from other servers and other users within the OSN and must be unique within a server.

The server stores a set of aliases for each user $a@X$. This set contains for each friend an entry with the alias $a@X$ is known by and the server of the friend $\{$(*alias@own_server*, *server_of_friend*),...$\}$. This way an authoritative server can resolve an incoming object addressed to an alias to the real username.

It is important to note that the alias sets do not contain any information about aliases of friends. The aliases of friends are only known by their own servers and by their friends.

Figure 2 shows an example of the local knowledge of two users $a@X$ and $b@Y$ and their respective servers X and Y. The example shows that only the two users know who the real usernames behind *both* aliases are. Even during communication between $a@X$ and $b@Y$, the servers X and Y are not able to determine any additional information about their users' friends, except that a user is communicating with *someone* at another server (noted as *?@server*).

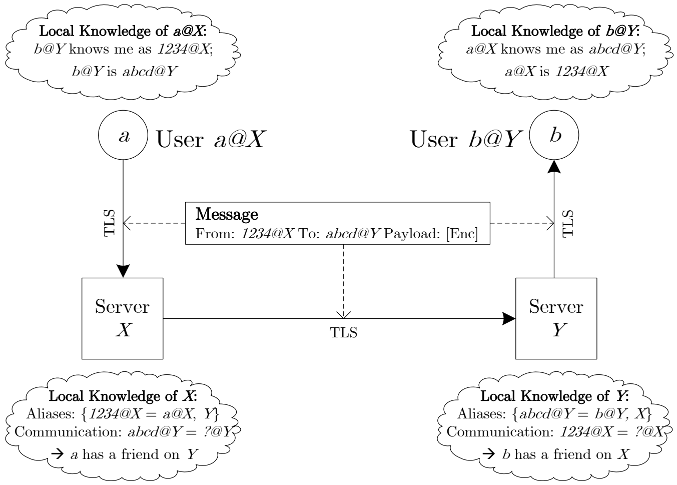

Figure 2. Disclosed information during communication

Since the aliases are unique for every friend, friends are unable to see which friends a friend has and if there are common friends, unless this information is intentionally disclosed by a user to his friends.

A. Assigning Aliases and Becoming Friends

The main problem when creating new friendships between users is to not disclose personal information during the procedure of becoming friends until both users have proved their identities. Thus to provide security and authenticity, a handshake has to take place which involves exchanging each other's public key. It is essential that no man-in-the-middle attack can be performed, e.g. if $a@X$ wants to establish friendship with $b@Y$, both server X and server Y could intercept messages and provide forged public keys to both clients. This problem of initial key exchange is well known and several solutions have been proposed (e.g. Diffie-Hellman key exchange).

To achieve authenticity and security between two users trying to establish a new friendship we propose two friendship establishment procedures. First, an *out of band* authentication using already established trusted connections between users (e.g. email, Skype, ...) and second, a modified version of the socialist millionaires' protocol [2].

Out of band: As shown in figure 3, before creating aliases the users $a@X$ and $b@Y$ have to exchange their usernames and public keys, using a third party communication channel (e.g. email). Afterwards the user $a@X$ initializes the alias creation leading to $a@X$ knowing an alias to contact $b@Y$ and vice versa. Using these aliases $a@X$ puts a friend-request object containing its signed real username and a se-

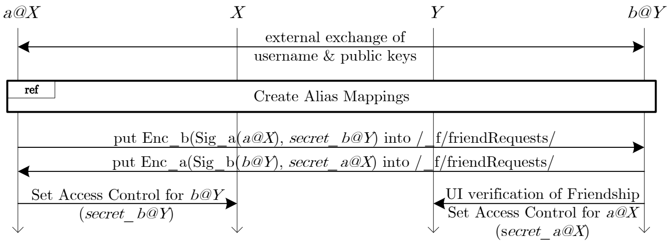
a@X
X
Y
b@Y
external exchange of
username & public keys
ref
Create Alias Mappings
put Enc_b(Sig_a(a@X), secret_b@Y) into /_f/friendRequests/
put Enc_a(Sig_b(b@Y), secret_a@X) into /_f/friendRequests/
Set Access Control for b@Y
(secret_b@Y)
UI verification of Friendship
Set Access Control for a@X
(secret_a@X)

Figure 3. Out of band friendship creation

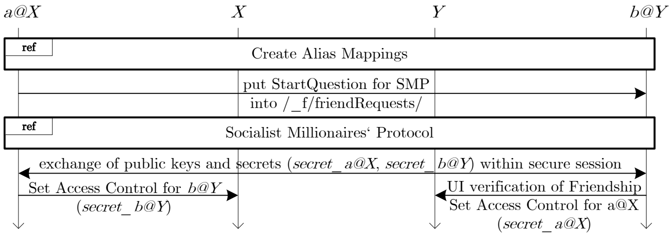
a@X
X
Y
b@Y
ref
Create Alias Mappings
put StartQuestion for SMP
into /_f/friendRequests/
ref
Socialist Millionaires‘ Protocol
exchange of public keys and secrets (secret_a@X, secret_b@Y) within secure session
Set Access Control for b@Y
(secret_b@Y)
UI verification of Friendship
Set Access Control for a@X
(secret_a@X)

Figure 4. Modified Socialist Millionaires' Protocol friendship creation

cret for calculating the message authentication code (MAC) encrypted into a specific directory /*b@Y*/_f/friendRequests/ of user *b@Y*. The user *b@Y* creates and stores the friend-request object the other way around. Now both users can verify the identity of the incoming friend-request and grant their new friend permission to write data objects.

Modified Socialist Millionaires' Protocol: For our second friendship establishment procedure we chose the socialist millionaires' protocol (SMP) for mutual authentication as shown in [2]. SMP is a zero knowledge protocol which allows two parties to compare a secret without revealing anything about that secret except whether their secrets are identical. We propose to use the SMP in such a way that the user *a@X* who sends a friendship request to another user *b@Y* only has to enter a question and a matching answer which is only known to him and his friend (see Fig. 4). If the protocol succeeds, both users end up with the correct public key of another. Using this authenticated public key *a@X* and *b@Y* can check the identity of each other and exchange the secrets for writing access verification.

Assigning aliases: When creating *single direction aliases* a user's identity needs only to be known to his own server and to his friends after the friendship was established. The identity or personal information do not need to be disclosed to other authorities, in particular not to the friends' servers. Figure 5 shows the alias creation in detail for a user *a@X* requesting an alias for addressing a user *b@Y*. The client of *a@X* requests a new unused alias from its own server *X*. Within this request *a@X* states that this alias is determined for *someone* at server *Y* (*?@Y*) and thus server *X* does not know who will use this alias to contact *a@X*. However, server *X* does know that *a@X* can be contacted under the specific alias from someone at server *Y*. First *a@X*

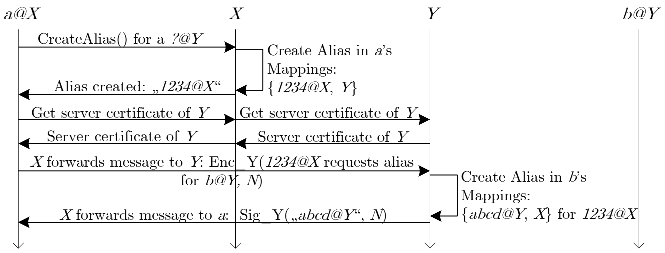

Figure 5. Creation of aliases

requests the server certificate of server *Y* to encrypt the communication between *a@X* and *Y*. By relying on X.509 *a@X* ensures that the certificate is valid and has not been tampered with. *a@X* requests a new alias for user *b@Y* on server *Y*. To hide its identity from server *Y*, *a@X* uses its own alias as sender that is intended for communication with *b@Y*. Furthermore, it contains a cryptographic nonce *N* to detect replay attacks. Server *Y* now creates the alias for *b@Y* and notes that the alias is used by *?@X*. Then server *Y* stores the requesting alias of *a@X* to notify *b@Y* how the user may contact *a@X*, even though neither the server *Y* nor *b@Y* know which user is related to this alias. However, *b@Y* will learn who the real user behind the alias is during the friendship creation before accepting the friendship request. In the end, server *Y* signs the created alias for *b@Y* and *N* and sends them back to server *X*. After *a@X* has validated the signature and checked *N* he knows an alias to contact *b@Y*, as well as the alias that *b@Y* will use to contact him. The user *b@Y* knows how to contact *a@X* and which alias *a@X* will use to contact him (after finishing one of the friendship creation protocols mentioned earlier). The server *X* only knows that *a@X* can be contacted using a specific alias by *?@Y* who uses another specific alias (and vice versa for server *Y*). Thus, the friendship relation between the real users cannot be recreated by one server alone.

B. Replication and Anonymous Retrieval

To increase availability of data objects we use caching on non-authoritative servers. This means that there can be several copies of a data object in the federation as a whole, since servers cache requested data objects that may be reused. If a user *a@X* wants to read the object /*b@Y*/_s/file1 and server *Y* is currently unreachable, server *X* first checks if it has a cached version. In this case the object will be retrievable without noticing the downtime of server *Y*. If this is not the case, *b@Y*'s friends could still have a cached version. However we do not want do disclose any information about friendship relationships and thus cannot directly tell *a@X* which friends of *b@Y* may have cached versions of the requested object.

A key feature of our obfuscated OSN is, that replicated or cached user-data at friend-servers can be retrieved without disclosing any friend-relationships, even to friends. Our approach is a primary copy replication scheme realized as follows: Clients must be able to determine 1) on which servers they can access replicated data and 2) under which alias. Thus every user stores his alias set (that is maintained by its server) encrypted in his storage under the path /_s/aliasset. This file has to be updated by the client periodically. Every friend fetches this set and thus gets the information of aliases for $b@Y$ at other servers without knowing which friends of $b@Y$ these aliases represent.

Whenever the server Y of a friend $b@Y$ is not responding a user $a@X$ can contact a server listed in the alias-set and request cached objects for a specific alias. The server holding the replicated data neither knows which user is requesting the objects, nor whose replicates are requested. The request itself does not have to be restricted since only friends of the author will be able to decrypt the data.

C. Implications for Access Control

Hiding the identity of users from other servers has implications for access control. Servers can't use the public keys of remote users to verify data objects. Instead a symmetric key is used to generate a MAC. Servers have a list which maps aliases to such a key. Incoming objects can be verified by servers without requiring knowledge about their authors' usernames.

However the server is not able to verify that within the encrypted content the valid real username is provided. This can only be checked by the client by decrypting the content and checking if it is signed correctly. If posts happen to be spam or contain an invalid signature the storage's owner is able to see the alias that has abused its writing permissions and revoke the permissions for this user.

D. Implications for Encryption

If an owner decrypts a comment posted at his storage, he will know the username belonging to the author's alias. Given this username and group information, he can choose the correct key.

If a client reads a comment on a remote storage, it has only the alias from the storage's owner point of view. It therefore cannot choose the correct key because the related mapping is not known to him. However, the group and author's host name can be used to reduce the set of possible authors. From this reduced set, tentative decryption is performed with each key. Once a correct key has been found, clients can store the deduced alias to username mapping to speed up future decryption. We chose this method of decryption to protect obfuscation. If key identifiers were unique, an attacker would be able to crawl hosts for encrypted circle keys. Once a key with an identifier I has been found in $a@X$'s storage, an attacker could deduce that all posts encrypted with I have been written by $a@X$. This is prevented by choosing colliding key identifiers.

IV. Security Assessment

In this section we will discuss the provided security of our approach. We considered active and passive attackers in different scenarios and evaluated both obfuscation and confidentiality.

A. Local Area Network

In this scenario an attacker Mallory resides in the same local area network (LAN) as $a@X$. Using eavesdropping Mallory cannot break confidentiality since $a@X$ establishes an encrypted connection to her server. Even if Mallory modifies, forges or drops packets confidentiality still holds due to the integrity provided by TLS. An attack during connection establishment will also fail due to server authentication.

Since Mallory cannot break confidentiality, breaking obfuscation using transmitted payload data is not possible. However, Mallory can observe communication patterns, which might reveal parts of the social graph. Considering only sender and receiver on a network layer level, there is no information leak since $a@X$ communicates with her server exclusively. Her server might forward data to her friends but since the attack takes place in the LAN Mallory also cannot observe how data from $a@X$ is forwarded by her server.

If $a@X$ and $b@Y$, one of her friends, are in the same LAN, Mallory could deduce that they are friends. Although both only exchange encrypted messages with their servers, the timing can be considered. If $a@X$ sends a chat message to $b@Y$, X will forward this data object to Y and Y will push it to $b@Y$. Although Mallory cannot get hold of the content, correlation between these roughly equally sized messages allows Mallory to assume a friendship relation between $a@X$ and $b@Y$ with a certain probability. Repetition of such patterns, e.g. in a chat session, can increase that probability. In this attack, Mallory can only observe network addresses of $a@X$ or $b@X$. She still does not know neither any alias of them nor their usernames.

Such attacks can be circumvented by sending bogus traffic and delaying message forwarding. However, such solutions decrease the efficiency. Especially with a mobile device, sending bogus traffic can require too much resources. Therefore, balancing the probability of such an attack with the costs of its mitigation, we accept this as a possible weakness.

B. Server

We will now assume that the attacker is a malicious server. As before, confidentiality remains unbroken. Only clients are in possession of the required keys. Social graph obfuscation can be reversed in a limited amount. Since servers have to keep a mapping of aliases to usernames, the malicious server can reconstruct all local friendship relations. However, this attack is limited to local friends only. Servers could also cooperate to break obfuscation between them. As before in the local case, this does not affect obfuscation of users on other servers, even if they are friends with a user on a malicious server.

Like Mallory in the previous scenario, servers can perform correlation attacks. If $a@X$ and $b@X$ are users of a malicious server, X could determine if they have a common friend $c@Y$. Although the alias for $c@Y$ is different for both $a@X$ and $b@X$, X can observe that $a@X$ and $b@X$ request identical objects from *?@Y*.

Besides passive attacks, the server could also perform data manipulations. Modifying existing data objects will cause clients to notice this since data objects are stored in signed containers. The keys for these containers are only known to clients and therefore servers cannot forge signatures. The same is true for creating new containers.

The only possible data manipulation attack which is not detectable by clients are data object deletes. Those could be mitigated by clients generating cryptographic proofs of nonexistence as in [3]. However, this puts additional load on clients (every data object creation/deletion has to update these proofs) and only provides little benefit: Deletions could only be detected by clients but not prevented.

Clients store their keys in an encrypted container on servers. A server could perform a brute-force or dictionary attack on this to acquire all keys. This is mitigated by clients requiring users to choose strong passwords.

C. Fake Profile

A common problem in social networks are fake profiles, i.e. impersonation attacks. To be a threat to both confidentiality or obfuscation, the user had to be added as a friend in the first place. In this process, the user's identity will be verified (see section III-A). For a faked profile, this verification will fail and therefore render impersonation attacks futile.

D. Malicious Friend

If a user $a@X$ has a malicious friend $m@X$ because a former accepted friend becomes evil, there are other attacks to consider.

Since $a@X$ never publishes her friend list to anybody even her friends cannot remove obfuscation using it. However, if $a@X$, $b@Y$ and $m@X$ are friends to each other, there is a high probability that messages are exchanged. If $b@Y$ comments one of $a@X$'s posts using a circle $m@X$ is part of, $m@X$ will know that they are friends. This could be mitigated by either making comments only visible for the original author of a post or by making posts anonymous. We believe however that this would decrease the benefits of an OSN since its functions would reduce to some kind of private messaging. Furthermore, this attack only succeeds if $m@X$ is a friend of $b@Y$. Otherwise $m@X$ would not be able to decrypt $b@Y$'s comment to $a@X$'s post.

Confidentiality, in its meaning of preventing unauthorized entities from acquiring secrets, is still given. This is because both $a@X$ and $b@Y$ gave $m@X$ permission to access their data and therefore $m@X$ is no longer an unauthorized entity. Other users of the OSN do not suffer from consequences of $a@X$'s and $b@Y$'s decision to become friends with $m@X$.

If $m@X$ and $b@Y$ are not friends, $m@X$ will not be able to deobfuscate $b@Y$'s identity, even if $b@Y$ comments on $a@X$'s posts. Since key identifiers collide for different users, $m@X$ cannot correlate these with users.

V. Cryptographic Performance

To verify the feasibility on mobile devices, we measured the performance of cryptographic operations on an average consumer mobile phone (HTC Desire S, Android 4.1.2) We chose RSA-2048, SHA-1 and RC4 as cryptographic primitives. We assumed an average user having 3 circles and 100 friends and calculated the mean value out of 100 test runs.

Before a user can participate in the OSN, he has to generate an asymmetric key pair. This one-time step required $3155.19\,\text{ms}$ ($\pm 2088\,\text{ms}$). Creating a posts consists of 3 key encryptions, encrypting the post, generating an RSA signature and calculating a MAC. This took $28.47\,\text{ms}$ ($\pm 1\,\text{ms}$) for a post of 150 bytes length.

Reading a post consists of choosing the correct key, RC4 decryption and RSA signature verification. In worst case, all 100 friends are in the same circle on the same host and the client has no known alias-to-user mappings. In this case, 100 keys have to be tested for decryption which took $10.44\,\text{ms}$ ($\pm 0\,\text{ms}$) in total.

If a user is removed from a circle, we have to generate a new circle key for all remaining users in this circle. Again we consider the worst case with 99 RSA encryption operations and generation of one RSA signature. This finished after $56.12\,\text{ms}$ ($\pm 1\,\text{ms}$).

VI. Related work

There have been several approaches to address the problem of privacy in OSN. Some approaches are peer-to-peer based, for example Safebook [4], LifeSocial [5], PeerSoN [6], porkut [7], Cachet [8] and My3 [9]. However, peer-to-peer solutions have some disadvantages. Nodes have a high fluctuation which can lead to data becoming unavailable or even lost. To prevent this, there is in most cases a DHT that assigns responsibility to another node once a node leaves the network. Therefore, if data is to be retrieved by a client, it has to find the node *currently* responsible for it. Depending on the actual overlay this can cost some time. Considering our requirements, those solutions may have efficiency issues (**R3**) since in a peer-to-peer network one has to keep connections to several other peers to speed up data lookups and maintain the network in general. This can require too much resources for mobile devices – in particular bandwidth and battery. You could create a bridge that provides a simple interface for mobile devices and maintains a connection to the peer-to-peer network, but this would introduce another point of failure.

Besides peer-to-peer networks, there have also been federated approaches before like OneSocialWeb [10] and Diaspora [11]. Both do not use end-to-end security and thus fail to comply with our privacy requirement **R1**; providers are able to read their users' data.

Other federated systems like [12] and [13] disclose the social graph of the user. While some users may find this acceptable, it provides a different level of privacy guarantee than our obfuscation method.

Another approach is the Vegas [14] which is also a federated system with end-to-end secure data. Vegas hides the social graph but contrary to our approach interaction between participants is limited to one-to-one chat messages. Status updates, a feature present in all major OSNs, could be implemented by broadcasting messages to all friends. However, a message requires two asymmetric cryptographic operations for each recipient, i.e. $2n$ operations for a message addressed to n users. As asymmetric cryptography is particularly expensive, in our system 1 asymmetric cryptographic operation is used for a message with any number of recipients. The correlation attacks discussed in Section IV applies to Vegas as well. An eavesdropper in a LAN can deduce whether two users in this LAN are friends to each other.

Vegas and also other systems like [13] do not use replication. If the storage of a user is offline, her profile information will be inaccessible by her friends. As explained in the introduction, we consider replication or caching to be crucial for a decentralized OSN to achieve a decent availability (**R2**).

VII. Conclusion

We proposed a federated OSN in which all content is end-to-end secure between user devices. Servers act as storages for encrypted data objects without having access to the keys. User-generated content can not be read or forged by the server providers at all, meeting our privacy requirement **R1**. To hide the social graph from the servers we use an obfuscation technique which maps a username to a different alias for each directed edge in the graph. This way server providers can only identify friend relationships on their own servers, or need to collude to disclose the social graph beyond server boundaries. The obfuscation approach is compatible with replication, improving the availability of data (**R2**) in case of temporary server downtimes. The application client always communicates directly with its own storage server, saving the overhead of many short-lived connections to third-parties. Despite using cryptograhic operations extensively, performance measurements suggest that the application runs efficiently on mobile devices, complying with our efficiency requirement **R3**.

References

[1] A. Bielenberg, L. Helm, A. Gentilucci, D. Stefanescu, and H. Zhang, "The growth of diaspora - a decentralized online social network in the wild," in *Computer Communications Workshops, 2012 IEEE Conference on*.

[2] C. Alexander and I. Goldberg, "Improved user authentication in off-the-record messaging," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*.

[3] B. Laurie, G. Sisson, R. Arends, and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence," RFC 5155 (Proposed Standard), IETF.

[4] L. Cutillo, R. Molva, and M. Onen, "Safebook: A distributed privacy preserving online social network," in *World of Wireless, Mobile and Multimedia Networks, 2011 IEEE International Symposium on a*.

[5] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz, "Lifesocial.kom: A secure and p2p-based solution for online social networks," in *Consumer Communications and Networking Conference, 2011 IEEE*.

[6] S. Buchegger, D. Schiöberg, L. H. Vu, and A. Datta, "PeerSoN: P2P social networking - early experiences and insights," in *Proceedings of the Second ACM Workshop on Social Network Systems Social Network Systems 2009, co-located with Eurosys 2009*.

[7] R. Narendula, T. Papaioannou, and K. Aberer, "Privacy-aware and highly-available osn profiles," in *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2010 19th IEEE International Workshop on*.

[8] S. Nilizadeh, S. Jahid, P. Mittal, N. Borisov, and A. Kapadia, "Cachet: a decentralized architecture for privacy preserving social networking with caching," in *CoNEXT*, 2012.

[9] R. Narendula, T. Papaioannou, and K. Aberer, "My3: A highly-available p2p-based online social network," in *Peer-to-Peer Computing, 2011 IEEE International Conference on*.

[10] Vodafone Group. (2011) Onesocialweb - creating a free, open, and decentralized social networking platform. [Online]. Available: http://onesocialweb.org/

[11] Diaspora Inc. (2012) https://joindiaspora.com/. [Online]. Available: http://onesocialweb.org/

[12] F. Raji, A. Miri, M. Jazi, and B. Malek, "Online social network with flexible and dynamic privacy policies," in *Computer Science and Software Engineering, 2011 CSI International Symposium on*.

[13] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*.

[14] M. Dürr, M. Maier, and F. Dorfmeister, "Vegas - a secure and privacy-preserving peer-to-peer online social network," in *Social Computing, 2012 IEEE Fourth International Conference on*.