Secret Interest Groups (SIGs) in Social Networks with an Implementation on Facebook

Alessandro Sorniotti
SAP Research and Institut Eurécom
Mougins, France
alessandro.sorniotti@eurecom.fr

Refik Molva
Institut Eurécom
Valbonne, France
refik.molva@eurecom.fr

ABSTRACT

In this paper we present the first framework that allows the creation of Secret Interest Groups (SIGs) in Online Social Networks; SIGs are self managed groups formed outside of the social network, around secret, sensitive or private topics. Members exchange credentials that can be used inside the social network to authenticate upon friendship requests or to secure user-generated content. To this end we present a set of cryptographic algorithms leveraging on well-studied primitives, and we describe a java implementation of the framework for Facebook.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

Keywords

Social Networks, Secret Interest Groups, Secret Handshakes

1. INTRODUCTION

Ever received a friendship request on Facebook reading "Hi, I'm John Smith, add me as a friend, we were classmates at university", remaining clueless about who this person is? Or ever been told that there is a profile under your name on a given social network, except you have never created such profile? If you are avid social network users, the answer to the first question is most likely yes, and a good percentage will answer yes to the second question too.

Indeed Online Social Networks (OSN) are becoming one of the most prominent communication technologies. Platforms as Facebook now count millions of users that are sharing information every day. Since the content is in many cases hosted on the OSN provider, OSN users can be profiled and offered detailed advertisement to support the OSN provider's revenues from advertisement.

A problem which is particularly felt among social network users is identity theft and identity spoofing [1, 2]. The root of the problem is that in many OSNs there is little or no verification that a person that joins the social network is really who he or she claims to be. This shortcoming needs to be combined to a second one: social network users base the decision on whether to accept a friendship request on name, pictures and fragments of text, information that is often easily retrievable elsewhere on the internet. It is therefore relatively simple [5] for an attacker to set-up a profile on an OSN, pretending to be somebody else, and then to convince other users to accept friendship request, consequently sharing their private information with the attacker.

To improve the security of this process, users could be asked to provide credentials along with the friendship request. However to be meaningful, credentials cannot be self-generated, but need to be generated and maintained after verification by a third party: this task is cumbersome and expensive and on the one hand, it is unrealistic to expect a central entity (the social network provider for example) to attend to it for free, and on the other hand, users are not likely to pay for such service.

A viable solution consists on users creating ad-hoc, trusted groups *outside of the social network*, issuing group membership credential and presenting such credentials upon friendship invitations within the social network.

A natural evolution of the aforementioned trusted friends groups are Secret Interest Groups (SIG), user-created groups with particular attention to confidential or simply privacy-sensitive topics. Indeed users of online social networks are often also exchanging personal and sensitive material; moreover, OSNs are more and more the theater of political, religious debate, often used as means to exchange confidential material that cannot go through official channels. It has been the case for instance in Iran during the recent post-electoral turmoils [3].

Our goal in this paper is therefore the creation of a framework that supports the formation and evolution of Secret Interest Groups. With our framework, users are able to handle the joining and leaving of members, to revoke or grant administration privilege to members, and to grant credentials that members can use to secure their relationship with other members in a social network. The challenges in the design of the framework are mainly two: it should (i) suit the requirements of ad-hoc groups, namely, it should not require a centralized entity but operate in a distributed fashion and (ii) it should be possible to implement it in a real OSN, with all the constraints thereof, for instance, impossibility of direct connections between users, all the communications going through OSN servers and the fact that users are not

SAC'10 March 22-26, 2010, Sierre, Switzerland.

necessarily online simultaneously.

The SIG framework that we have *designed and implemented*, supports the aforementioned technical requirements along with the high security requirements typical of secret groups. These groups are secret in the sense that membership to a SIG is a sensitive information that users are reluctant to expose publicly. A SIG can be for instance centered around religious, political, sexual interests, such that users are very interested to interact with other users that share the same interest, but extremely reluctant to admit publicly that they belong to such an interest group. This notwithstanding, a SIG can be centered around less secret topics, that still represent privacy-sensitive topics requiring a certain degree of security or privacy. The proposed framework therefore addresses the more complex case of secret groups, but can be used for simple private user groups too.

Our contribution In this paper we first study the operational and security requirements of a generic SIG framework; then, we propose the complete set of cryptographic algorithms and protocols that describe the operations of our SIG framework, respecting the aforementioned requirements. Last, we have implemented the SIG framework to make it usable in the ever growing OSN platform Facebook. To the best of our knowledge ours represents the first effort in the field.

In the design of the cryptographic part of the solution, as we shall see later, we have relied on Secret Handshakes [4] as one of the main pillars. From this perspective, our work also represents one of the first realistic use cases for secret handshakes. In addition and even more interestingly, the Secret Handshake scheme that we propose is the first one to relax the need for a centralized entity to issue credentials; finally, it is the first secret handshake scheme to use signatures and Oblivious Signature-Based Envelopes as credentials (although the possibility of using them has been hinted in [16]).

2. DESIGN OF THE SIG FRAMEWORK

The SIG framework can be split into two main parts: *OSN external* and *OSN internal*. As the names suggest, the main difference is that some of the algorithms of the SIG framework will operate outside of the social network, independently of it, while other algorithms will be executed in the social network, using the social network itself as a transport, in order to secure further communications among the social network users.

OSN external algorithms deal with the creation and maintenance of the Secret Interest Group outside of the social network: as real friendship happens outside of the social network and is then used to create some links inside of it, a secret group is created outside of the social network and it is used to secure friendship links established within. *OSN internal* algorithms instead deal with authentication, handshaking and encryption of content among users of the social network, using the social network as a transport layer for cryptographic messages.

2.1 OSN external

All OSN external algorithms must take into account that the SIG is an ad-hoc group. We will therefore adopt three design guidelines: (i) there should be no central entity, (ii) any entitled user should be able to act as a central entity and (iii) algorithms should be "thresholdized", in the sense that to be performed successfully they need the cooperation of at least a minimum number of entitled users. Respecting these guidelines results in the role of the central entity being split and spread among entitled users.

We assume that the creation of a new SIG group is the task of an initial set of *SIG managers*, that creates the group and handles the joining of the first *SIG members*. The initial set of SIG managers may reserve the ability of handling the joining of other users to themselves only, or may endorse other users with this capability as well. The difference between SIG managers and SIG members is substantially that SIG managers are normal SIG members who are also responsible for appointing new SIG managers and to allow new members to join the SIG. From this, we derive the first two requirements of SIG Join:

- **RExt1**: at any point in time, the set of SIG managers must be non-empty;
- **RExt2**: only a subset of SIG managers can appoint new SIG managers;

To enhance the security of SIGs, we require that one SIG manager alone is not able to perform either tasks: both procedures require instead a minimum number of SIG managers to be involved in their execution.

- **RExt3**: appointing new SIG managers and handling the joining of new members are distributed tasks, that no SIG manager alone can handle; instead, a joint effort of a minimum number of t SIG managers is required;

Prior to be allowed to join the SIG, a user must undergo offline verification carried out by a SIG manager; the purpose of this verification is to ensure that all members are consistent with the SIG join policy. This procedure is group-specific and requires different controls depending on the nature of the SIG: for example, in a SIG revolving around political militancy, party membership, background check or face-to-face interview could be the check that a user is required to pass before being admitted in the group; for a SIG representing a project consortium, it may be instead sufficient to send the membership credentials using the consortium email address. The same must be true for SIG managers.

- **RExt4**: SIG managers will admit new SIG members or new SIG managers only after checking their compliance to the SIG join policy;

SIG members and SIG managers receive membership credentials and managership credentials to certify their roles. An additional requirement aims at making sure that these credentials cannot be forged or modified:

- **RExt5**: no coalition of less then t SIG members or SIG managers is able to forge a new credential (both membership or managership);

The existence of credentials comes along with the requirement for revocation. We will treat the revocation of SIG managership credential differently from simple SIG membership ones. There are mainly two well-known techniques to revoke credentials: proactive and reactive techniques. The first embeds time-limits in the credentials, or forces periodic updates of the credentials. The second singles out revoked credentials publishing a revocation handle to an authenticated public list.

The fact that single SIG manager credentials fall into the hand of an attacker is not so dangerous, thanks to requirement **RExt3**: an attacker would indeed need to get hold

of at least t SIG managership credentials, and since t is a parameter of the system, this threshold can be set based to meet the desired degree of security. The most suitable solution to counter the theft of SIG manager credentials is therefore a proactive strategy: managership credentials can be periodically updated so as to make sure that the probability that an attacker gets hold of at least t valid managership credentials is arbitrarily low.

As for SIG membership credentials, revocation is required when either a SIG member got his membership token stolen or when he no longer qualifies for membership. The loss of a SIG membership credential is somewhat thornier then the loss of SIG managership credentials, since SIG membership credentials can be used directly to authenticate to another SIG member or to access content of a SIG member, as we shall see in the next Section; therefore a reactive revocation approach is required. Promptly detecting that a credential has fallen in the hands of an attacker is a vital requirement: for stolen credentials this is quite straightforward, since the legitimate user that suffered the theft can realize it and report it (the situation is a bit more complex in case credentials are stolen due to a virus/trojan compromising the system but leaving the attacker unaware, but this aspect is out of our scope). Much more complex is the case of detecting a once legitimate user betraying his allegiance to the SIG and colluding with an attacker, or becoming one himself: however this aspect is orthogonal to our work and represents a separate area of research. We therefore assume that the following assumption holds:

- **REext6**: stolen SIG membership credentials or credentials belonging to a user that has become malicious are eventually detected as such;

2.2 OSN internal

A SIG member eventually wants to add another alleged SIG member to his list of friends through the standard social network invitation process, exchange content or chat in a secure way, both operations that occur within the social network. Two main algorithms are therefore required: mutual authentication of two SIG members and encryption of content for fellow SIG members. The first requirement is then straightforward:

- **RInt1**: only a legitimate SIG member can successfully authenticate to another SIG member or receive content from the latter;

Keeping in mind that SIGs are by definition secret or privacy sensitive, the invitation process is crucial, because a legitimate member (the inviter, the invitee or both) does not yet know whom he is interacting with. Indeed, sending a friendship request on the grounds of common SIG membership would imply admitting to belong to the SIG for the inviter; accepting the request would mean the same for the invitee. Inviter and invitee have no incentive in disclosing their SIG membership unless they can be sure that they are interacting with another SIG member.

Such authentication problem can be solved by Secret Handshakes. Secret Handshakes have first been introduced in 2003 by Balfanz et al. [4] as mechanisms designed to prove group membership between fellow group members. Additional properties are that non-members must not be able to either impersonate group members or to recognize legitimate group members. Besides, the communications between group members are designed so as to provide untraceability of any two protocol exchanges. The purpose of these protocols is to model in cryptography the folklore of real handshakes between members of exclusive societies, or guilds. Upon handshake between two SIG members, due to the nature of SIGs, users are reluctant to disclose their affiliation to the SIG: they want to do so just when they are sure to be interacting to another legitimate SIG member. From this observation, we derive the following requirement:

- **RInt2**: when two OSN users are trying to authenticate as SIG members, either both learn that they both belong to the SIG or they do not learn anything at all;

This implies that proof of membership and verification of membership happen simultaneously and atomically. Revocation should respect this requirement, in that revocation of a SIG membership credential should forbid its owner to both prove or verify membership to the SIG.

2.3 Security and Adversarial Model

There are mainly four different actors in our SIG framework: the OSN provider, the OSN user who is not a SIG member, the OSN user who is a simple SIG member and finally the OSN user who is a SIG manager. Each of these actors have different goals and different capabilities.

The OSN provider can be modeled as a Dolev-Yao type of intruder: although unlikely to meddle with the users' content, the OSN provider effectively controls the network and in theory has the possibility to read, modify and drop each and every message that is being exchanged in the OSN. In reality, the OSN provider is more likely to behave as a Honest-But-Curious adversary, whose goal is to perform data-mining and to profile users so as to gather data that – depending on the regulations of the country where the OSN provider has its legal body – can be sold to third parties, or to simply use this information to offer specific ads to users thus gaining revenues from advertising. Seen from a SIG point of view, the OSN provider is interested in passive, monitoring attacks, such as discovering whether a user is part of a SIG, in discovering the nature of a SIG and the type of content that is being protected before exchange among SIG members. The OSN provider can of course spawn fake OSN users, but cannot become a SIG member since SIG membership is given only after compliance check with the SIG join policy (**REext4**). Under the assumption of perfect cryptography, the main goals of this type of attacker are therefore linkability and traceability; linkability refers to the ability of linking different users to the same SIG, therefore create a list of SIG members; traceability refers instead to the ability of tracing the same member over multiple executions of SIG related operations.

Simple OSN users can also be modeled as Dolev-Yao intruders for the following reason: the OSN provider (a Dolev-Yao attacker by definition) can both spawn OSN users or collude with real OSN users. This type of intruder's objectives are similar to the OSN provider, with the addition of active attacks, where the attacker engages in the authentication protocol with legitimate SIG users with the objective of passing off as a legitimate SIG user.

A SIG manager/member has already access to the information a simple OSN user/OSN provider is after. Under the assumption that the revocation mechanisms hold, a SIG member/manager that loses or maliciously gives away credentials is eventually caught. The goal of such type of attacker is therefore to generate fresh SIG membership cre-

dentials to circumvent revocation, or to appoint new SIG managers without the consent of a majority of managers.

3. THE SIG FRAMEWORK

In order to build a scheme that satisfies the aforementioned requirements, we leverage on a number of well established cryptographic primitives. First, we use mechanisms for Secret Sharing and Secret Redistributions to allow SIG managers to share a secret and perform joint computations. Then, we leverage on threshold signatures to allow a set of SIG managers to jointly compute a signature. The signature will be the membership token allowing its possessor to prove its membership to the group. Finally, we use Oblivious Signature-Based Envelopes (OSBEs) to allow two users, owning a signature computed as discussed above, to perform a secret handshake and share a key iff they both own the signature of the same message under the same public key. In the next section we briefly illustrate these three techniques.

3.1 Preliminaries

At first, let us introduce some terminology that will be used in the rest of the section. Let p and q be large prime numbers such that q divides $p-1$ and let g be a generator of the subgroup of order q of $\mathbb{Z}_p$; let h be a one-way hash function in the range $\{1,\ldots,q-1\}$.

Secret sharing allows a set of n parties to possess shares of a secret value, such that any t shares can be used to reconstruct the secret, yet any $t-1$ shares provide no information about the secret. Secret sharing was first proposed by Shamir [22] and independently by Blakley [6]. This initial idea has been extended in a number of works [12, 13, 14, 20, 19]. For the objectives of our scheme, we will use two different algorithms: Share [20, 19], used by n users to share a random secret without a dealer, so that t are in principles able to reconstruct it; and Redistribute [12], used by t shareholders to compute n' new, unrelated shares of the same secret, so that t' new shareholders can reconstruct the secret. In both algorithms, the secret is actually never reconstructed, and – since there is no dealer – none of the shareholders knows the secret. We call $\Gamma_{\mathcal{P}}^{(n,t)}$ the access structure wherein a secret is shared among a population of users in the set $\mathcal{P}$, with $|\mathcal{P}|=n$ so that any subset $\mathcal{B}\in\Gamma_{\mathcal{P}}^{(n,t)}$, with $|\mathcal{B}|=t$ can reconstruct that secret.

Share : this algorithm is executed by each P_i belonging to an authorised set $\mathcal{B}\in\Gamma_{\mathcal{P}}^{(n,t)}$ with cardinality t. Each P_i picks a random $r_i \overset{R}{\leftarrow} \mathbb{Z}_q$, forms a random polynomial $f_i(u)=r_i+a_{i,1}u+\ldots+a_{i,t-1}u^{t-1}$ and sends $f_i(j) \mod q$ to each $P_j\in\mathcal{P}/P_i$; the (unknown) shared secret is $R=\sum_{j\in\mathcal{P}} r_j$. Every $P_i\in\mathcal{P}$ computes its share of the secret $R_i=\sum_{j\in\mathcal{B}} f_j(i)$; additionally, each P_i broadcasts $g^{r_i} \mod p$; this way, everybody can compute $g^R \mod p$;

Redistribute : this algorithm is executed by each P_i belonging to an authorized set $\mathcal{B}\in\Gamma_{\mathcal{P}}^{(n,t)}$ with cardinality t. The objective is to generate new shares for the new access structure $\Gamma_{\mathcal{P}'}^{(n',t')}$. Each P_i computes a random polynomial formed as $f_i(u)=R_i+a_{i,1}u+\ldots+a_{i,t'-1}u^{t'-1}$, where R_i is the local share of the secret possessed by P_i; each P_i sends $f_i(j) \mod q$ to each $P_j\in\mathcal{P}'/P_i$; then, each P_i can locally generate its new share R_i' by Lagrange interpolation;

Using these two algorithms, we can generate threshold signatures. At first we describe a variant [18] of the famous DSS signature scheme [9]. Let the secret signing key be $x\in\mathbb{Z}_q$ and the public key be $g^x \mod p$. The scheme has two algorithms:

Sign : given the message m and a random number $e \overset{R}{\leftarrow} \mathbb{Z}_q$, compute the signature (w,v) such that $w=(g^e \mod p) \mod q$ and $v=wx+h(m)e \mod q$;

Verify : (w,v) is a valid signature on the message m iff $w=\left(g^{vh(m)^{-1}}(g^x)^{-wh(m)^{-1}} \mod p\right) \mod q$;

A threshold signature scheme [8, 10, 11] leverages on the aforementioned secret sharing techniques in order to share the secret key among n parties, thus distributing the signing capabilities over n parties, so that any subset of t can jointly compute a signature, whereas no $t-1$ subset can. In [18], Park and Kurosawa propose a threshold version of the aforementioned DSS signature variant. The variant assumes that the Share algorithm has been executed to create an access structure $\Gamma_{\mathcal{P}}^{(n,t)}$ and that any principal in $\mathcal{P}$ has a share x_i of the (unknown) private key x. In addition, the public key g^x is publicly known. The Verify algorithm stays the same, whereas the Sign algorithm is modified as follows:

Sign : this algorithm is executed by each P_i belonging to an authorized set $\mathcal{B}\in\Gamma_{\mathcal{P}}^{(n,t)}$ with cardinality t; each P_i has a local share x_i of the secret signing key x. All the P_i engage in the Share algorithm, generating the value $g^e \mod q$ and a local share e_i of the (unknown) random value e; then, each P_i sends the value $v_i=g^e x_i+h(m)e_i \mod q$ to the requester of the signature, along with the value $g^e \mod q$; the first part of the signature is $w=g^e \mod q$; given the set of shares $\{v_i, i\in\mathcal{B}\}$, the second part of the signature v can be computed through Lagrange interpolation;

As we can see, the Share algorithm is executed twice, once prior to signing to generate the public key and shares of the private key, and a second time, to generate the first part of the signature and shares of its discrete log.

Finally, we introduce Oblivious Signature-Based Envelopes (OSBEs) as our building block for our OSN internal algorithms. OSBEs have been introduced by Li, Du and Boneh in [15]. An OSBE scheme allows two parties to share a key iff a predefined party among the two possesses a signature on an agreed-upon message. Nasserian and Tsudik have presented – among others – an OSBE scheme [16] based on the DSS variant mentioned in the previous section.

At first a message m is chosen; the OSBE round happens between a party P1 who might have a signature (w,v) on m and P2. The OSBE round proceeds as follows:

OSBERound : P1 sends w to P2; P2 generates $r \overset{R}{\leftarrow} \mathbb{Z}_q$, sends g^r to P1 and computes $K_2=\left((g^x)^w w^{h(m)}\right)^r$; P1 computes $K_1=(g^r)^v$; $K_1=K_2$, i.e. P1 and P2 will share a key iff P1's signature on m was correct;

3.2 Putting it all together

In this Section we describe our SIG framework based on its algorithms. At first let us describe three OSN external algorithms that are executed to populate and manage the SIG. As we have already mentioned in the previous sections, there are two different types of SIG users: regular members and initiators/managers; the following algorithms focus in issuing credentials for these two roles:

InitiateSIG : this algorithm is executed by a set of n SIG initiators; each SIG initiator jointly engages in the Share algorithm, forming the access structure $\Gamma_{\mathcal{P}}^{(n,t)}$, computing the SIG public key $PK_{\mathcal{SIG}} = g^x$ and shares of the private key $SK_{\mathcal{SIG}} = x$ that we represent as $SK^i_{\mathcal{SIG}} = x_i$; the public parameters are the SIG public key $PK_{\mathcal{SIG}} = g^x$ and a message $M_{\mathcal{SIG}}$; $SK^i_{\mathcal{SIG}}$ represents the SIG managership credential;

UpdateSIGManagers : this algorithm is executed by a set of t SIG managers; each SIG manager jointly engages in the Redistribute algorithm, forming a new access structure $\Gamma_{\mathcal{P}'}^{(n',t')}$ computing new shares of the private key $SK^i_{\mathcal{SIG}} = x_i$ and giving them to the new set of SIG managers $\mathcal{P}'$;

GrantSIGMembership : this algorithm is executed by a set of t SIG managers out of the n total; each SIG manager checks the requesting users' conformity to the SIG join policy; after a successful check, each of the t SIG managers engages in the Sign algorithm, forming a new signature on message $M_{\mathcal{SIG}}$, that verifies correctly under the SIG public key $PK_{\mathcal{SIG}}$; the signature is then issued to the supplicant user; the signature represents the SIG membership credential;

The SIG starts with the invocation of InitiateSIG by the initial set of n SIG managers, the SIG initiators. At the end of the algorithm, the SIG managers have picked a message that will be used later on for the handshake between two users on the social network, they have agreed on a known public key and they have distributed secret shares of an (unknown) private key, representing the SIG managership token. Notice that the actual private key is unknown; it could be known if at least t SIG managers colluded and reciprocally revealed their shares, however the threshold t can be set accordingly in order to discourage such attempts. In situations where hierarchy of SIG managers is important, each SIG managers can be supplied a different number of shares according to their importance.

The algorithm UpdateSIGManagers can be invoked by at least t SIG managers to redistribute *different* shares of the *same* SIG private key to a *different* set of SIG managers, fixing a new threshold t'. The reasons for invoking the UpdateSIGManagers are mainly twofold: (i) if the group grows, new SIG managers need to be appointed and therefore receive shares $SK^i_{\mathcal{SIG}}$ of the private key; (ii) when a group of (less then t) SIG managers needs to be revoked[1], t other SIG managers can generate a different set of shares, not related with the old set, and distribute the new shares to all managers but the ones whose manager rights need to be revoked, thus effectively preventing the latter from further executing SIG manager tasks. Regular invocations of UpdateSIGManagers can be scheduled so that new shares are proactively being distributed and the population of SIG managers can be both purged of elements that are no longer trustworthy and enriched with new trusted ones.

t SIG managers can also issue SIG membership tokens, that users can use to authenticate on the social network. To this end, t SIG managers execute GrantSIGMembership, issuing to the new SIG member a signature representing the SIG membership token.

[1]Notice that revoking SIG managership to t SIG managers or more is not feasible since in principles they can reconstruct the SIG secret key $SK_{\mathcal{SIG}}$; this can be countered by appropriately setting t.

U_1	$\longrightarrow$	U_2 $\quad w_1 = g^{e_1}, g^{r_1}$
U_2	$\longrightarrow$	U_1 $\quad g^{r_2}, w_2 = g^{e_2}$
U_1		computes $K_1 = \left((g^x)^{w_2} \, w_2^{h(M_{\mathcal{SIG}})} \right)^{r_1}$
		$K'_1 = (g^{r_2})^{v_1}$ and $k_1 = H(K_1 \| K'_1)$
U_2		computes $K'_2 = \left((g^x)^{w_1} \, w_1^{h(M_{\mathcal{SIG}})} \right)^{r_2}$
		$K_2 = (g^{r_1})^{v_2}$ and $k_2 = H(K_2 \| K'_2)$
U_2	$\longrightarrow$	U_1 $\quad E_{k_2}(N)$
U_1	$\longrightarrow$	U_2 $\quad E_{k_1}(N+1)$

Figure 1: SIGMembersHandshake and relative challenge-response upon friendship invitation.

Once group members receive their membership tokens, they can interact more securely on the social network platform, using the following algorithms:

EncryptForSIGMember : this algorithm is executed by a SIG member (sender) that wants to encrypt some content prior to its publication on the social network platform, to be received by a second SIG member (receiver); receiver and sender engage in the OSBERound algorithms, acting as P1 and P2 respectively; output of OSBERound is a key that the sender will use to encrypt the content before its publication on the social network. Only if the receiver is a legitimate SIG member, will he be able to reconstruct the key and therefore decrypt the content of the message;

SIGMembersHandshake : this algorithm is executed by two SIG members that want to authenticate one another as members of the SIG; the two members engage in two separate instances of the OSBERound algorithms, acting in turn as both P1 and P2 over the two executions, at the end of which, each party obtains two keys; the two parties then have to prove one another knowledge of both keys simultaneously; if this is successful, the Handshake is successful;

EncryptForSIGMember can be used for instance when trying to send a message or to post some content that is stored in the OSN servers and displayed to the recipient upon its access to the OSN; SIGMembersHandshake instead is the ideal candidate for securing the friendship invitation process in the OSN, but can also be used to secure synchronous events like chat sessions. For clarity's sake, SIGMembersHandshake is summarized in Figure 1: two users, U_1 (holding the signature (w_1, v_1)) and U_2 (holding the signature (w_2, v_2)) engage in two OSBERound sessions establishing two keys each. Then, they engage in a challenge-response protocol to prove to one another knowledge of the keys computed: as an example for the latter, we have decided to use a challenge-response protocol similar to the one used in Kerberos [17], with the addition that E is an authenticated encryption mode of operation for cryptographic block ciphers, such as OCB [21] (Offset Codebook Mode): this way U_1, upon receipt of the last message, is already able to tell whether he is interacting with a legitimate SIG member.

Of crucial importance is the possibility to revoke SIG membership tokens. There are two possible ways in which this can be achieved: one, suggested by Boneh and Franklin in [7], consists in periodically updating $M_{\mathcal{SIG}}$. For example, concatenating the current year to $M_{\mathcal{SIG}}$ = "abc" would make sure that only SIG members owning a signature on the message "abc || 2009" are able to successfully perform EncryptForSIGMember and SIGMembersHandshake. At the

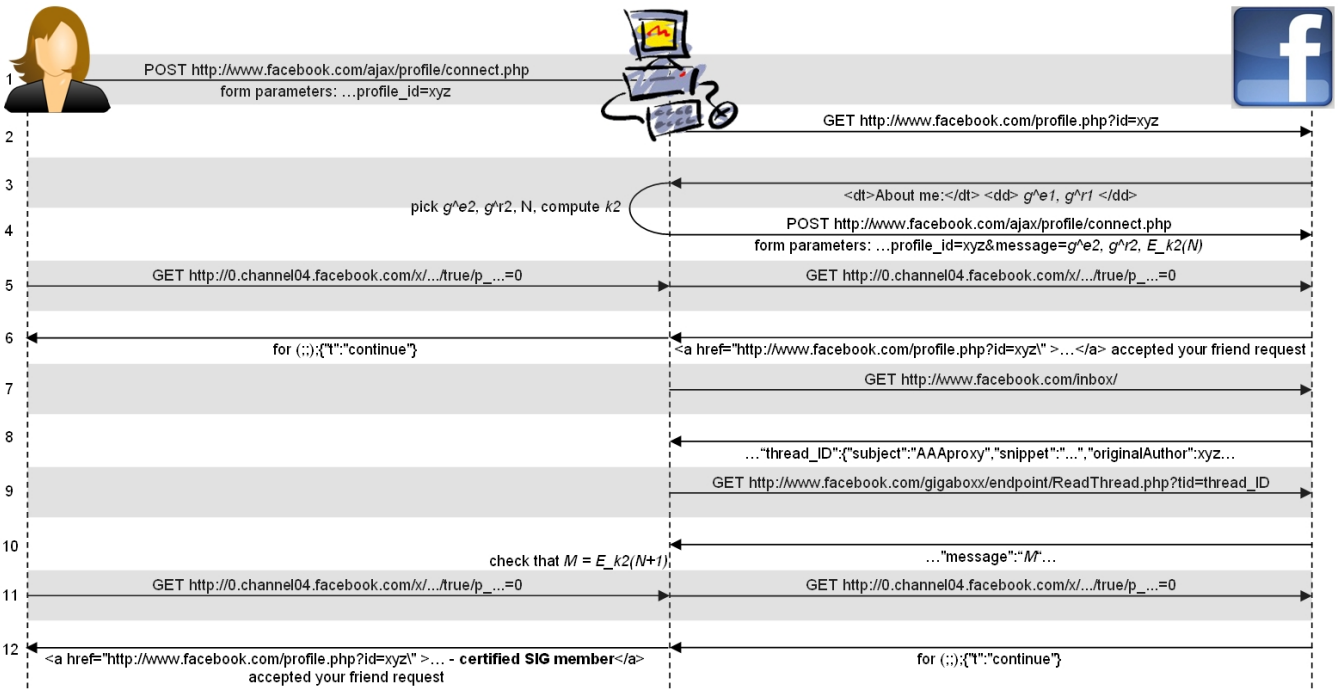

Figure 2: Operations of the proxy upon a friendship request.

expiration date, a SIG member simply needs to apply for GrantSIGMembership and, if it is still eligible, get a new signature for the next time period, in the example, for the next year.

The second approach for revocation is a reactive one, similar to the one proposed in [4]. This approach is based on the fact that upon execution of both EncryptForSIGMember and SIGMembersHandshake algorithms, users exchange part of their SIG membership credential, in particular, the first component w of their signature (w, v). w is formed by the t SIG managers that engage in the GrantSIGMembership, who know this value. Consequently, should this credential be revoked, the t SIG managers can just forge a new signature on the message "w `is a revoked SIG membership token`" and broadcast it to all SIG members. This signature can be posted by any means chosen by the users so as to speed up its diffusion, as it does not represent secret information.

4. IMPLEMENTATION IN FACEBOOK

In this Section we describe the implementation of a working proof-of-concept of the SIG framework. We have chosen to focus our implementation efforts only to the OSN internal part, and have picked Facebook as our target OSN. These choices are motivated by the fact that the implementation of the OSN external algorithms does not raise any interesting challenge; the choice of Facebook as OSN platform is motivated by its extreme popularity, which could ease the adoption and diffusion of our SIG framework.

The OSN internal part of our SIG framework has been implemented as a java http proxy. The intended use is the following: a SIG member runs the proxy, which intercepts only requests toward Facebook servers. The proxy modifies requests and responses, running the secret handshake protocol upon membership invitation and chat events; notification of the success or failure of the protocol is provided to the user through modifications of the html that is displayed in the browser. Among the features that still need to be implemented, on which we are concentrating our implementation efforts, the two most prominent ones are the encryption of messages and the porting of the software to a standard Facebook application.

The main challenge to face when implementing the OSN internal algorithms in a real social network, is that users of a social network are not always online, and cannot communicate directly, whilst the secret handshake protocol – for instance – is an interactive protocol. The challenge is therefore to adapt an interactive protocol to a non-interactive environment. In the rest of this Section we will therefore describe in details how the proxy operates upon a friendship invitation event at both inviter and invitee's side.

In Figure 2 we can see how the proxy operates upon a friendship request, triggered by message 1. The message is not forwarded immediately to the Facebook servers; instead the proxy looks up the profile of the invitee, extracting from the "About me" field of the profile the invitee's handshake (messages 2 and 3). Then the proxy creates its own handshake message, derives the key which is used to encrypt a random nonce N. Handshake message and encrypted nonce are then serialized and base64'ed, and added as the POST parameter "message" of message 1. The resulting message is finally forwarded to the Facebook servers (message 4). Let us assume that the invitee eventually accepts the request; the inviter is notified in a number of ways depending on whether he is online or not at the moment of the acceptance: the proxy intercepts this event in all its possible forms; as an example, in message 5 and 6, the browser is notified through the response to an infinite javascript loop that originates AJAX requests to fetch the updates. As can be seen, the message confirmation is not sent back directly

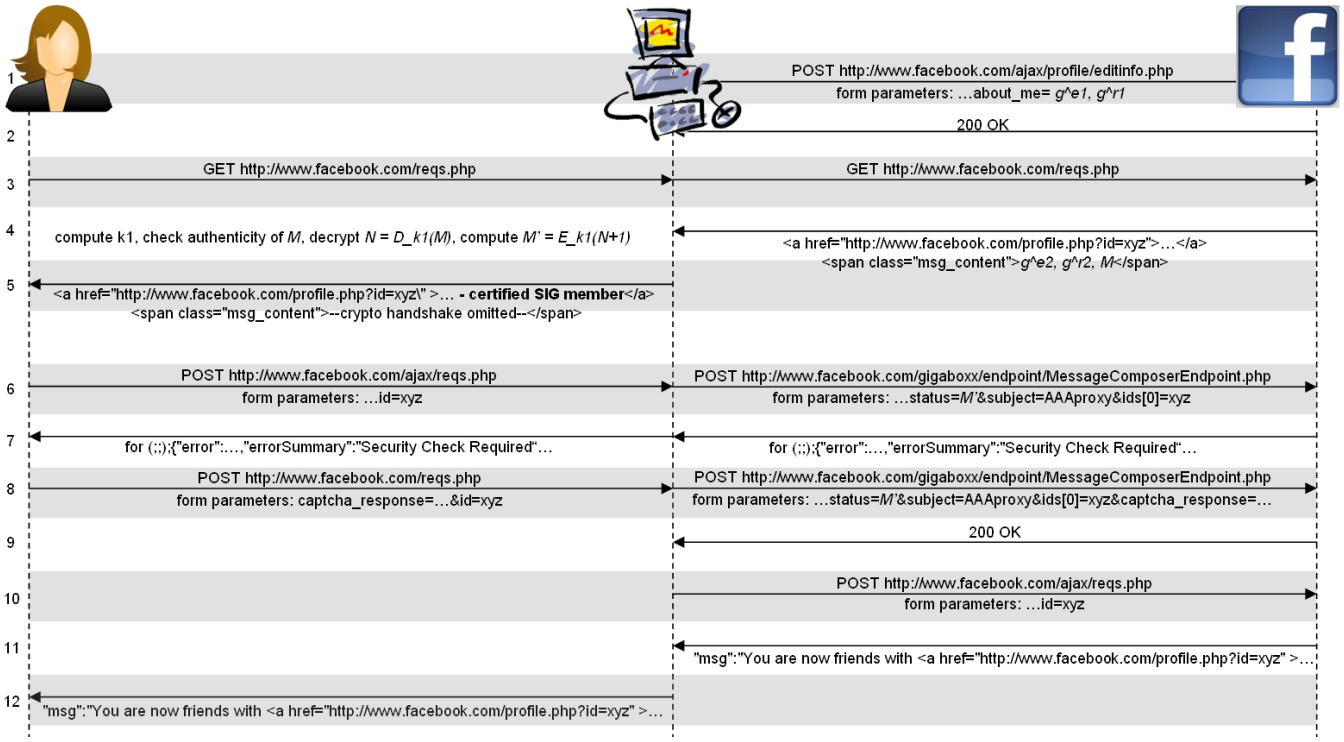

Figure 3: Operations of the proxy upon a friendship response.

to the browser. Instead, the proxy searches in the Facebook inbox for a message from the invitee with the response to the challenge (messages 7 to 10). If none is found or the message (after base64 decoding and deserialization) cannot be decrypted to be $N+1$, then the standard acceptance of message 6 is forwarded back to the client; otherwise a modified confirmation message that highlights the SIG membership of the invitee, like the one of message 12, is fed back to the browser.

Figure 3 shows instead how the proxy operates on the invitee's side. At first, the proxy publishes the invitee's handshake message in the "About me" section of the invitee's profile (messages 1 and 2). The operations begin with the client visiting the page with the pending requests (message 3). The page is fetched and the message that the inviter has included in the invitation is decoded and deserialized to the handshake message and the inviter's challenge M. Then the proxy, using the handshake message that had been serialized an base64'ed in the "About me" section of the profile (message 4), derives the key and attempts the decryption of M into N; since the encryption scheme – AES OCB – provides authenticity, a successful decryption implies that the inviter is a certified SIG member. In this case, the html that is sent back to the client (message 5) is modified so as to notify that one of the inviters is a SIG member. At this point, the user may decide to accept a friendship request. If so, the acceptance message is not forwarded right away; before it, a Facebook message is composed, with the string "AAAProxy" as subject, and with the encryption of $N+1$ in the body (message 6). Facebook uses captcha to prevent proxies like ours to perform automated actions (message 7); the proxy cannot clearly solve the captcha challenge, but can nevertheless forward the request back to the client to have it solved instead. The message, with the solution to the captcha is then sent to the Facebook server (message 8). Upon receipt of the confirmation (message 9), the proxy finally accepts the original friendship request and forwards the response of the server back to the client (messages 10 to 12). Right after the friendship acceptance has been sent, the steps of messages 1 and 2 are repeated, thus creating a new handshake message to be used for a new request. An arbitration protocol needs to be devised in case two invitations use the same handshake message.

5. SECURITY ANALYSIS

Due to space restrictions, we are only able to sketch the security analysis, since we have focused mainly on the gathering of the requirements and on the design and implementation of the framework.

The framework has been built intentionally on well-studied cryptographic primitives, in order to leverage on their security to guarantee that of the whole framework. In Section 2.3 we have identified three main typologies of adversary among the four actors of the framework: passive adversary (tracing and linking), active outsider (engaging in authentication without credential) and active insider (engaging in authentication with credential bypassing revocation).

Let us start with the requirements of untraceability and unlinkability in the context of passive adversaries; both algorithms EncryptForSIGMember and SIGMemberHandshake provide unlinkability, since the only group-wide value x, present in public key, private key and its shares, is only used in the computation of the keys output of the handshake; the keys are randomized, and only ciphertext encrypted with them is actually transmitted: assuming that a ciphertext does not reveal anything about the key used for encryption,

both protocols are unlinkable. Untraceability can easily be achieved by providing users with a number of one-time-use signature pairs (w, v). w is indeed the randomizer for the signature: since protocol Share guarantees that w is chosen at random in presence of an honest majority (and there are alternative protocols in case of dishonest players), then both protocols provide untraceability too. However, once protocols are implemented in the OSN, the requirements become harder to satisfy: in particular, untraceability is unpractical to be achieved since OSN users are tagged with OSN profile ID and profile names, that can be used for tracing. Notice that situations in which it is theoretically possible to provide untraceability, but then practical implementations provide tracing facilities at lower layers (the OSN in our case) happens frequently in security. Unlinkability instead is preserved: provided that there is a significant number of other SIGs in the OSN, it is not possible to link users to a given SIG.

Active outsiders' goal is to authenticate as a SIG member without owning a membership token. Such an adversary has a number of options to achieve the goal: tricking SIG managers into issuing them a SIG membership token is not feasible, since it would require tricking more then t managers, whereas t can be made big enough to counter such attempts; collecting t shares of the private key is not feasible either for the same reason and also due to the fact that shares can be proactively revoked; finally, SIG membership tokens are DSS signatures, whose security has been studied extensively in the literature [9]; this signature scheme guarantees resistance to existential forgery, therefore an active outsider has no possibility of forging membership tokens. Without a valid credential, users cannot engage in successful authentication; this follows from the property of semantic security against the receiver, which is guaranteed by OSBEs [16].

Active insiders have similar goals as active outsiders, with the difference that they dispose of a number of valid credentials; this fact can be modeled with an oracle that generates a number of signatures on $M_{\mathcal{SIG}}$, with the attacker then trying to generate a new signature with a different randomizer (thus circumventing revocation). However, the same considerations mentioned for active outsiders apply here: resistance to existential forgery includes this oracle, and therefore its existence does not impact the security of the scheme; the same applies for the semantic security against the receiver of OSBEs.

6. CONCLUSION AND FUTURE WORK

In this paper we have focused our attention on the problem of securing user interaction in online social network, mainly through the creation of self-managed user groups that hand out credentials to their members. Then, secret-handshake based authentication and content encryption are used in the social network. To this end we have defined a set of requirements, sketched a security model, presented a framework of cryptographic protocols and introduced a proof-of-concept java implementation, working in the ever growing Facebook platform.

This paper leaves two major items for future work: first of all, a more thorough security analysis is required, although its bases have been sketched in Section 5. Finally, the java prototype should be extended to become an actual Facebook application, to support all the functionalities of the framework, and to be usable in other OSN platforms.

7. ACKNOWLEDGEMENTS

This work has been partially supported by the SOCIALNETS project, grant agreement number 217141, funded by the EC seventh framework programme.

8. REFERENCES

[1] `http://cryptoblog.wordpress.com/2009/07/08/social-networks-and-social-security-numbers/`.

[2] `http://chris.pirillo.com/pownce-social-networks-arent-identity-networks/`.

[3] `http://www.guardian.co.uk/world/2009/jun/22/neda-soltani-death-iran`.

[4] D. Balfanz, G. Durfee, N. Shankar, D. K. Smetters, J. Staddon, and H.-C. Wong. Secret handshakes from pairing-based key agreements. In *IEEE Symposium on Security and Privacy*, 2003.

[5] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All your contacts are belong to us: automated identity theft attacks on social networks. In *WWW*, 2009.

[6] G. Blakley. Safeguarding cryptographic keys. In *AFIPS Conference Proceedings*, volume 48, pages 313–317, 1979.

[7] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3), 2003.

[8] C. Boyd. Digital multisignatures. Cryptography and Coding, 1986.

[9] R. H. Brown and A. Prabhakar. Digital signature standard (dss).

[10] Y. Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO*, pages 120–127, 1987.

[11] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO*, 1989.

[12] Y. Desmedt and S. Jajodia. Redistributing secret shares to new access structures and its applications, 1997.

[13] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, 1987.

[14] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO*, 1995.

[15] N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. *Distributed Computing*, 17(4):293–302, 2005.

[16] S. Nasserian and G. Tsudik. Revisiting oblivious signature-based envelopes. In *Financial Cryptography*, pages 221–235, 2006.

[17] B. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, 32(9), Sep 1994.

[18] C. Park and K. Kurosawa. New elgamal type threshold digital signature scheme. 1996.

[19] T. P. Pedersen. Distributed provers with applications to undeniable signatures. In *EUROCRYPT*, 1991.

[20] T. P. Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT*, volume 547, pages 522–526. Springer-Verlag, 1991.

[21] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. Ocb: A block-cipher mode of operation for efficient authenticated encryption. 2001.

[22] A. Shamir. How to share a secret. *Commun. ACM*, 22(11), 1979.