Event Invitations in Privacy-Preserving DOSNs
Formalization and Protocol Design

Guillermo Rodríguez-Cano, Benjamin Greschbach, and Sonja Buchegger

KTH Royal Institute of Technology
School of Computer Science and Communication
Stockholm, Sweden
{gurc,bgre,buc}@csc.kth.se

Abstract. Online Social Networks (OSNs) have an infamous history of privacy and security issues. One approach to avoid the massive collection of sensitive data of all users at a central point is a decentralized architecture.
An event invitation feature – allowing a user to create an event and invite other users who then can confirm their attendance – is part of the standard functionality of OSNs. We formalize security and privacy properties of such a feature like allowing different types of information related to the event (e. g., how many people are invited/attending, who is invited/attending) to be shared with different groups of users (e. g., only invited/attending users).
Implementing this feature in a Privacy-Preserving Decentralized Online Social Network is non-trivial because there is no fully trusted broker to guarantee fairness to all parties involved. We propose a secure decentralized protocol for implementing this feature, using tools such as storage location indirection, ciphertext inferences and a disclose-secret-if-committed mechanism, derived from standard cryptographic primitives. The results can be applied in the context of Privacy-Preserving DOSNs, but might also be useful in other domains that need mechanisms for cooperation and coordination, e. g., Collaborative Working Environment and the corresponding collaborative-specific tools, i. e., groupware, or Computer-Supported Collaborative Learning.

Keywords: Event Invitation, Privacy, Decentralized Online Social Networks

1 Introduction

The most common form of Online Social Networks (OSNs) are run in a logically centralized manner (although often physically distributed), where the provider operating the service acts as a communication channel between the individuals. Due to the popularity of these services, the extent of information the providers oversee is vast and covers a large portion of the population. Moreover, the collection of new types of sensitive information from each individual simply keeps increasing [19]. Users of these centralized services not only risk their own privacy

but also the privacy of those they engage with. Whether intentional, or unintentional, data leakages [18], misuse [13] or censorship are some of the issues affecting the users.

Decentralization has been proposed to reduce the effect of these privacy threats by removing the central provider and its ability to collect and mine the data uploaded by the users as well as behavioral data. A Decentralized Online Social Network (DOSN) should provide the same features as those offered in centralized OSNs and at the same time it must preserve the privacy of the user in this different scenario. The latter is not straightforward, as in addition to the decentralization challenge itself, new privacy threats arise when the gatekeeper functionality of the provider that protects users from each other disappears [8].

One of the standard features of OSNs is the handling of event invitations and participation, i. e., a call for an assembly of individuals in the social graph for a particular purpose, e. g., a birthday celebration, demonstration, or meeting. There is usually metadata related to each event, such as date, location and a description. An implementation of this feature must provide security properties to the participants, e. g., that a user can verify that an invitation she received was actually sent by the organizer. Furthermore, it must support certain privacy settings. For example, an organizer could choose that only invited users learn how many other users were invited and that only after a user has committed to attend the event, she learns the identities of these other invited users.

Realizing this in a decentralized scenario is non-trivial because there is no Trusted Third Party (TTP) which all involved users can rely on. This is a problem, especially for privacy properties where information shall only be disclosed to users with a certain status, because any user should be able to verify the results to detect any possible cheating. In the example above, a neutral, trusted broker could keep the secret information (the identities of invited users) and disclose it only to users who committed to attend the event. This would guarantee fairness to both the organizer and the invited users. It becomes more challenging to implement this without a central TTP and still allowing different types of information about the event to be shared with different groups of users in a secure way.

1.1 Our contribution

We describe and formally define two basic and five more complex security and privacy properties for the event invitations feature.

We propose and discuss a distributed and privacy-preserving implementation of the event invitations feature without using a TTP. The suggested protocols cover all of our defined properties, considering 20 different parameter combinations for the tunable privacy properties.

We also describe three privacy-enhancing tools that we use in our implementation: storage location indirection, controlled ciphertext inference and a commit-disclose protocol. They are based on standard cryptographic techniques such as public key encryption, digital signatures and cryptographic hashes, and can be useful for other applications as well.

1.2 Paper Outline

We discuss related work in Section 2, describe the problem of implementing the event invitation feature in a decentralized way and formalize security and privacy properties in Section 3. Our proposed implementation together with privacy-enhancing tools follow in Section 4, and we discuss this solution in Section 5. We conclude with a summary and future work in Section 6.

2 Related work

Groupware tools have been widely researched since they were first defined in 1978 by Peter and Trudy Johnson-Lenz [10]. Choosing between centralized and distributed implementations has been a major concern for these applications as pointed out in [15]. While the traditional model uses the client-server architecture [20, 12], there have been some projects on decentralized collaborative environments: Peer-to-pEer COLlaborative Environment [4], a P2P multicast overlay for multimedia collaboration in real-time, although synchronous; YCab [2], a mobile collaborative system designed for wireless ad-hoc networks; or a hybrid P2P architecture with centralized personal and group media tools in [21].

Security features in collaborative applications were already introduced in the popular client-server platform for businesses, IBM Notes/Domino (formerly Lotus Notes/Domino), to allow for usable authentication, and digital signature and encryption by means of a Public Key Infrastructure (PKI) to end-users [22]. Control policies in Computer-Supported Collaborative Work (CSCW) are considered in [16], including distributed architectures.

Protocol design guidelines in collaboration scenarios, where the privacy of a group member does not lessen by participating in the environment, have been studied and proposed in [11]. These guidelines aim at minimizing the amount of information a member has to provide to the group for the common activities, and making the protocols and the tasks transparent to everyone in the group.

Another type of related work lies within the domain of DOSNs [1, 3, 5]. To the best of our knowledge the event invitations feature has not been investigated in a privacy-preserving manner in this decentralized scenario.

3 Decentralizing The Event Invitation Feature

We already described the intuition of an event, where a group of people gathers with the intention of carrying out some activity. Now we more formally model the event invitation feature and desirable security and privacy properties. We denote the set of users as $U = u_1, \ldots, u_n$. The event invitation happens in three main stages:

- **Creation:** When a user $u_i \in U$ decides to create a new event e_k, she becomes the organizer o_{e_k} and creates the event object $event_k$ including different information, e. g., a description, date, time and location.

- **Invitation:** The organizer o_{e_k} selects the set of users to be invited to the event e_k, denoted by I_{e_k}, crafts the invitation objects $i_{e_k}^{u_j}$ for each of these invitees, and sends them to the respective users.
- **Commitment:** The invitees I_{e_k} have the chance of confirming the invitation, i. e., "commit" to attend the event e_k, by issuing commitment objects $c_{e_k}^{u_j}$. We denote the set of all attendees, i. e., the users who committed to the event e_k, as C_{e_k}.

Figure 1 shows an example with eight users, $u_1 \dots u_8$, where one of them, u_1, is the responsible organizer o_{e_k} of the event e_k. The organizer issues invitations to $u_2 \dots u_6$, depicted with a dashed line. These users form the group of invitees, denoted with I_{e_k}. Invited users who confirm their attendance, (u_2, u_4 and u_6 in this example), provide a commitment to the organizer, depicted with a continuous line. They form the group of attendees, denoted with C_{e_k}.

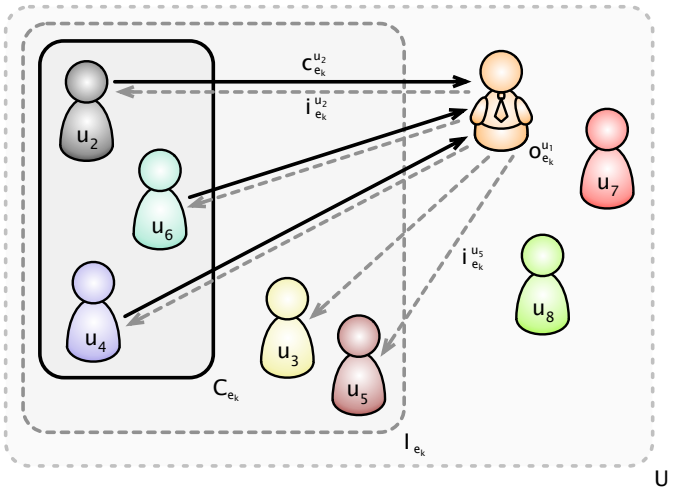

Fig. 1. Example of one event invitation.

A possible privacy setting could specify that invited users learn how many other users are invited but only attending users learn their identities. That is, u_3 and u_5 would learn that five users are invited (while this is kept secret from u_7 and u_8). u_2, u_4 and u_6 would additionally learn the identities of $I_{e_k} = u_2 \dots u_6$.

3.1 System Model and Assumptions

In the following, we assume basic functionalities of popular OSNs to be available in a decentralized manner, such as user search [9] and user messaging [17]. We also assume that users are identified by a public key and the ability to verify the identity of other users via some sort of PKI, which can be realized in a

decentralized manner, e. g., a "Web of Trust" model or a Bitcoin block-chain binding friendly usernames to public keys [6]. Moreover, we rely on a distributed storage featuring access right management, e. g., that a certain storage object is only writeable by a specific user, and "append-only" storage objects, where new data can be appended, but existing data cannot be modified or removed without notice. The latter can be realized in a decentralized fashion, e. g., in a similar manner as the Bitcoin block-chain is secured against modifications [14].

3.2 Threat Model

We assume that users in all roles, e. g., invited users or the organizer of an event, might act maliciously, i. e., become adversaries. The capabilities of an adversary range from passively learning information accessible in that role (e. g., an invited user might have access to a list of all other invited users, depending on the privacy settings for the event), to actively interacting with other parties, e. g., writing arbitrary data to accessible storage objects or sending arbitrary messages to other users. We also assume that powerful adversaries might have the possibility to pervasively monitor a large fraction of the network traffic. While we try to mitigate threats like traffic analysis and correlation attacks arising from this, we cannot completely protect against them and come back to this in the discussion section. We do not assume that adversaries can subvert the storage layer. So we assume the availability of a secure distributed storage including features like append-only lists and authorization mechanisms, as mentioned above.

We want to keep malicious users from undermining the reliability of the event invitation feature for legitimate users. This means that an adversary should not be able to violate the security and privacy properties that we define in the next section. This comprises guaranteeing the authenticity and non-repudiation of statements made by the involved parties, such as issued invitations or commitments. Furthermore it includes keeping information such as the identities of invited/attending users, the number of invited/attending users or a private event description secret from unauthorized users while guaranteeing its availability and authenticity for legitimate users. An example for the latter would be to keep an organizer from withholding or lying about the number of attending users. We do not focus on denial-of-service attacks and leave them for future work.

3.3 Security and Privacy Properties

A protocol for event invitations can comply with different security and privacy properties. We first list the following basic security properties:

- *A user u_j can prove that she was invited to the event e_k if and only if the organizer o_{e_k} invited u_j, i. e., issued an invitation $i_{e_k}^{u_j}$.*
  This property is two-sided and guarantees that a user cannot forge an invitation she did not get, while an organizer cannot deny that she invited a user. This implies that an invitation $i_{e_k}^{u_j}$ is tied to a user u_j that was chosen by the organizer o_{e_k} and cannot be transferred to another user.

– *An organizer o_{e_k} can prove that the invited user u_j committed to attend the event e_k if and only if u_j actually committed, i. e., issued a commitment $c_{e_k}^{u_j}$.* This property also has two sides. The organizer cannot forge a commitment of a user that did not commit to the event. And a user cannot deny that she committed to an event once she did so.

More challenging properties are those defining which groups of users are allowed to see what information, namely,

Invitee Identity Privacy (IIP)

For an event e_k, only a chosen set of users (e. g., U, I_{e_k}, C_{e_k} or only o_{e_k}) learns who else is invited (i. e., sees all members of I_{e_k}).

This property defines who can see information about who is invited to an event. This can be all users (U) or be restricted so that only other invited users see who else is invited (I_{e_k}). Another possibility is that even an invited user first learns who else is invited when she committed to attend (C_{e_k}). Finally, this information could be kept completely secret, so only the organizer o_{e_k} knows the complete list of invited users.

Invitee Count Privacy (ICP)

For an event e_k, only a chosen set of users (e. g., U, I_{e_k}, C_{e_k} or only o_{e_k}) learns how many users are invited (i. e., learns $|I_{e_k}|$).

This property is a variant of property IIP where the number of the invited people I_{e_k} is disclosed to a set of users (while the identities of the invited people might remain hidden).

Property IIP and ICP are closely related in the sense that if IIP holds for a certain set of users, then ICP trivially holds for the same set (and all its subsets – note the subset relation of the possible sets to choose from, $U \supseteq I_{e_k} \supseteq C_{e_k}$).

This constrains the possible combinations of these two properties' parameters. If, for example, for a certain event all invited users I_{e_k} should see who else was invited, i. e., property IIP with parameter choice I_{e_k}, then it does not make sense to choose that only the attendees C_{e_k} should learn the number of invited people, i. e., property ICP with parameter choice C_{e_k}, because the invited users can already derive this information from what they learn from property IIP.

Attendee Identity Privacy (AIP)

For an event e_k, only a chosen set of users (e. g., U, I_{e_k}, C_{e_k} or only o_{e_k}) learns who is attending (i. e., sees all members of C_{e_k}).

Attendee Count Privacy (ACP)

For an event e_k, only a chosen set of users (e. g., U, I_{e_k}, C_{e_k} or only o_{e_k}) learns how many users are attending (i. e., learns $|C_{e_k}|$).

Similarly to properties IIP and ICP, these two properties specify who can see information about the users who committed to attend an event. Property AIP defines who can see the identities of the attendees while property ACP defines to whom the number of attendees is disclosed. The same relation, regarding the possible parameter choices, as described for properties IIP and ICP, also holds here.

Attendee-only Information Reliability (AIR)

An invited user u_j can only get access to the private description $d^S_{e_k}$ of the event e_k once committed and the organizer o_{e_k} can only claim the attendance of the user u_j once the private description $d^S_{e_k}$ is available to u_j.

This property has two sides. First, a user u_j can only get access to information exclusive to the attendees C_{e_k}, i. e., the private description $d^S_{e_k}$ from the organizer o_{e_k} for an event e_k, if she has committed to attend. Second, and conversely, the organizer o_{e_k} can only claim that user u_j has committed to attend if she has made it possible for u_j to access the private description $d^S_{e_k}$.

4 Implementation

We now propose an implementation of the event invitation feature described in Section 3 in a privacy-preserving DOSN. We assume that user identifiers u_i are public keys, and we will denote their corresponding private keys as u_i^S (where S stands for "secret").

4.1 System Components

The main components of the system are event objects, invitation objects and commitment objects as depicted in Figure 2.

- **Event object:** When a user wants to create a new event, she first generates a public/private keypair e_k/e_k^S. The public key will become the identifier for the event and the user will be denoted as organizer o_{e_k}. She then assembles the event object $event_k$: She writes a public event description d_{e_k} and a private description $d^S_{e_k}$ that will be encrypted with a symmetric key PDK. She creates one list to store the invitation objects (*invite-list*) encrypted with a symmetric key ILK, another list for the commitment objects (*commit-list*) and one for disclosing secret information to committed users (*disclose-list*). The event object contains links ILL, CLL and DLL, pointing to the storage

locations of these three lists. Additionally the organizer creates a list of public/private keypairs $rk_1/rk_1^S, \ldots, rk_n/rk_n^S$, to encrypt the entries on the commit-list, and includes the public keys in the event object. Moreover, the event object contains information about the chosen privacy settings.
The organizer signs the public key of the event with her own user key to confirm that she is the organizer and signs the whole event object $event_k$ with the event's private key e_k^S. Therefore, an event object is composed as follows:

$$event_k = Sign_{e_k^S}(Sign_{u_i^S}(e_k)||u_i||d_{e_k}||Enc_{PDK}(d_{e_k}^S) \\ ||ILL||ILK||CLL||DLL||rk_1, \ldots, rk_n||\text{privacy settings})$$

Some of the elements of the event object might, however, be encrypted with additional keys or only be hashes (made with a cryptographic hash function H, e. g., SHA-2 [7]) of the actual values. This depends on the chosen privacy settings and will be explained in more detail later.

- **Invitation object:** An invitation object is composed of the invitee's identifier u_j (her public key), signed by the organizer o_{e_k} with the event's private key e_k^S:

$$i_{e_k}^{u_j} = Sign_{e_k^S}(u_j)$$

- **Commitment object:** A commitment object is composed of the invitation object $i_{e_k}^{u_j}$ and the cryptographic hash of the event object $event_k$, both signed by the attending user u_j with her private key u_j^S as follows,

$$c_{e_k}^{u_j} = Sign_{u_j^S}(H(event_k)||i_{e_k}^{u_j})$$

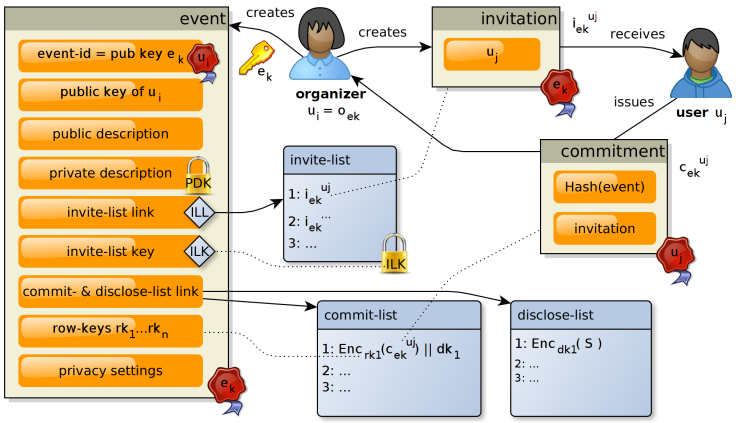

Fig. 2. Overview of the actors, system components and their relations.

4.2 Privacy Enhancing Tools

Before describing the implementation, we introduce tools that we will use several times.

Storage Location Indirection and Controlled Ciphertext Inference If we want to make the size of a list, i. e., the number of its elements, available to a subset of users, but not the content of the list elements (in our scenario because each element contains a user identifiers), we can use storage location indirection and ciphertext inference: The list will not be stored together with the event object, but at a secret location in the distributed storage such that it can only be reached if the link to it is known. Additionally, the elements of the list will be encrypted so that the stored content can only be accessed if the encryption key is known.

This provides the possibility of a controlled information disclosure depending on the knowledge of a user: Users who do not know the link, learn nothing, neither the size nor the content of the list. Making the link to the list but not the encryption key available to a subset of users, enables these users to learn the size of the list (assuming a constant ciphertext size for each entry), while it does not give them any details about the contents stored. Users that received both the link and the encryption key, learn the content and can act as verifiers, checking that there are no invalid entries that incorrectly increase the perceived number of elements as seen by those users holding only the link but not the key.

Commit-Disclose Protocol The organizer may want to share some information only with users who have committed to attend the event (attendees). To ensure fairness, the invited users need some guarantee that they can expect to receive the promised information when they commit to attend.

While this is easy to solve if both parties, the organizer and the invited users, trust a neutral third party that can act as broker, it becomes more difficult in our setting where we do not assume the existence of any TTP. So we base our solution on a significantly weaker trust assumption: the availability of append-only storage objects as described in Section 3.1.

The aim of the protocol is to provide an invitee u_j who commits to the event e_k with a secret S held by the organizer o_{e_k}. It is composed of three main components, provided by the organizer of the event:

- **Commit-List**, a public and append-only storage object where invited users store their (encrypted) commitments.
- **Disclose-List**, a public readable, but only writeable by the organizer, append-only storage object where the organizer discloses (encrypted) secrets for the committed users.
- **Anchor Point**, a storage object (in our case the event object) serving as common entry point, referencing the commit-list and the disclose-list either directly by providing their storage locations, i. e., a commit-list link CLL

and a disclose-list link DLL or indirectly by holding salted hashes of these storage locations (where DLL and CLL together with the salts are shared with a subset of users in another way). Additionally, a list of public keys $rk_1, \ldots, rk_n$, called row-keys, used to encrypt the entries on the commit-list are also stored here. All this information is signed by the organizer.

Each key in the row-keys list is intended for encrypting one entry of the commit-list. The corresponding private keys $rk_1^S, \ldots, rk_n^S$, are held by the organizer. The protocol runs in three phases:

- **Commit Phase:** If the user u_j wants to commit to attend the event e_k, she looks up the commit-list and finds the next free row – let this have index l. She then looks up the corresponding row key rk_l in the event object.
  Finally, she crafts a commitment $c_{e_k}^{u_j}$, creates a fresh keypair dk_l^P/dk_l^S (disclose key, later used by the organizer to encrypt the secret information) and writes the following entry to row l of the commit-list: $Enc_{rk_l}(c_{e_k}^{u_j})||dk_l^P$ that is the commitment, encrypted with the row-key, together with the public disclose key in plain.

- **Disclose Phase:** When the organizer o_{e_k} sees that a new row l has been added to the commit-list, she tries to decrypt the first entry, using the secret row key rk_l^S. If this succeeds and the commitment is valid the organizer writes the secret information, encrypted with the provided disclose key to row l of the disclose-list, i. e., $Enc_{dk_l^P}(S)$. If the decryption fails or the commitment is invalid, the organizer publishes the secret row-key of row l in the disclose-list instead, i. e., rk_l^S, thus proving to everybody who can access the lists that she was not obliged to disclose the secret information to the creator of row l.

- **Blame Phase:** If the organizer misbehaves and does not provide a protocol-abiding user with the secret information after a reasonable amount of time, the user can blame the organizer. She does this by publishing a blame-entry in the commit-list, referring to the row l and disclosing the secret disclosure key dk_l^S. Thus everybody who can access the lists can see that she did not receive the secret information encrypted to the disclosure key she provided in row l. It can be assumed that the commitment (which cannot be decrypted by the verifying public) was correct, as otherwise the organizer would have published the secret row-key of row l.

In this way, the commit-disclose protocol does not keep the organizer from cheating, but it allows the user to reliably blame the organizer if it is the case.

4.3 Basic Security Properties

The basic security properties are fulfilled by the construction of an event, invitations and commitments described in Section 4.1 and the guarantees of the

PKI. The first basic security property is fulfilled because an invitation $i_{e_k}^{u_j}$ for a user u_j is created by using the event's private key e_k^S, owned by the organizer o_{e_k} to sign the invited user's identifier. The invitee cannot forge the event's key and the organizer cannot deny having issued the invitation because the signature used to sign the invitation is publicly verifiable. The second basic security property is also fulfilled because an organizer o_{e_k} cannot forge a commitment $c_{e_k}^{u_j}$ as she is not able to forge another users' signature. A user u_j, having sent the commitment $c_{e_k}^{u_j}$ to the organizer o_{e_k}, cannot deny the commitment as her signature is again publicly verifiable and binding to the event e_k.

4.4 Invitee Identity Privacy and Invitee Count Privacy

In order to implement properties IIP and ICP, we let the organizer o_{e_k} store all invitation objects for the event in the invite-list. Retrieving the list requires knowledge of the invite-list link ILL, and in order to decrypt it, the symmetric invite-list key ILK must be known beforehand.

Knowledge of the link ILL is equivalent to learning the total number of invitations, even if the decryption key ILK is unknown because the number of invitations can be inferred from the size of the ciphertext in the list. Knowledge of the encryption key ILK allows learning the identities of the invited users I_{e_k} because the invitations $i_{e_k}^{u_j}$ store the user identifiers in plain text.

If the organizer o_{e_k} wants to make the identifiers of the invitees I_{e_k}, or the amount of them, i. e., $|I_{e_k}|$, available to all users U, she will publish ILL or ILK in plain text together with the event object $event_k$. Making this information available only for invitees I_{e_k} can be realized by the organizer privately sharing it with the invited users. In order to share the decryption key ILK only with the committed users C_{e_k}, the commit-disclose protocol can be used, while the link ILL is then either available publicly (i. e., choosing U for property ICP), shared only with the invitees (i. e., choosing I_{e_k} for ICP) or kept secret and only shared with the committed users together with ILK (i. e., choosing C_{e_k} for ICP).

It is also possible to avoid sharing any information about the invitations by keeping ILL and ILK secret, i. e., choosing o_{e_k} both for properties IIP and ICP. When the identities should not be known to anyone but the number of invitees should be made public to a subset of users (i. e., choosing o_{e_k} for property IIP), the link ILL will be shared with the respective users and a particular encryption scheme for the invite-list is employed: Instead of encrypting the invite-list as a whole, we encrypt its individual entries with the public keys of the recipient of the invitation stored at each entry. Thus, the invited users can verify that their own invitation is included in the list. However, this only allows for a weak verification of the correctness of the list, i. e., it provides an upper-bound of the size of the list, because the organizer o_{e_k} can add invalid or dummy entries (e. g., to artificially increase the perceived number of invitees to the event).

A summary of how ILL and ILK are shared depending on the choice of parameters for properties IIP and ICP is shown in Table 1. Note that the row describing the privacy settings IIP: C_{e_k}, ICP: I_{e_k}corresponds to the example mentioned in the introduction and Section 3.

Table 1. Sharing of ILL and ILK as per the IIP and ICP settings. P = publicly available in $event_k$, I = privately shared with I_{e_k}, C = shared only with C_{e_k} (via the commit-disclose protocol), S = fully secret (only o_{e_k} knows about it) and S* = special encryption scheme for the invite-list.

Settings		Implementation	
IIP	**ICP**	***ILL***	***ILK***
U	U	P	P
I_{e_k}	U	P	I
	I_{e_k}	I	I
C_{e_k}	U	P	C
	I_{e_k}	I	C
	C_{e_k}	C	C
o_{e_k}	U	P	S*
	I_{e_k}	I	S*
	C_{e_k}	C	S*
	o_{e_k}	S	S

4.5 Attendee Identity Privacy and Attendee Count Privacy

To implement the AIP and ACP properties, we mainly use the commit-disclose protocol. The link to the commit-list CLL can be shared publicly in the event object $event_k$ except for those cases where the count of attendees $|C_{e_k}|$ must be kept private. In this situation, if the invitees I_{e_k} are allowed to learn $|C_{e_k}|$, CLL is shared privately with them. Alternatively, the organizer can add dummy entries in the list to hinder inferences from the number of (encrypted) entries. When not even attendees should learn how many other users are attending, dummy entries in the commit-list are the only solution as the CLL must always be shared with all invitees, so that they can commit if they want to attend.

Dummy entries follow the pattern of usual entries, i. e., random data with a specific size to fake an encrypted commitment object and a public key in the commit-list, and random data in the disclose-list to fake an encrypted secret. All users who hold the private row-keys can identify them because the first part of a dummy entry in the commit-list cannot be decrypted with the respective row-key, while those users without the private row-keys cannot distinguish dummy entries from real ones as the ciphertext structure looks the same for all of them.

When the link CLL should not be shared publicly in the event object $event_k$, a salted hash of the link will be stored instead so that the organizer o_{e_k} cannot cheat by sharing different links with different groups of users. As the event object is unique per event and group of invitees, the invited users can check they all got the same link from the organizer by comparing it with the hash value in $event_k$.

Otherwise the implementation varies only in how the private row-keys are disclosed, as they protect the commitments in the commit-list: If all users U are allowed to learn who is attending, the private row-keys will be public, i. e., the rows do not need to be encrypted. If only the invited users I_{e_k} should see the

identities of the attendees, the private row-keys will be shared with the invitees directly. And if only the attending users should learn about the identities of other attendees, the private row-keys are disclosed using the commit-disclose protocol.

This way we are able to implement all possible parameter combinations of the AIP and ACP properties, except for the combination AIP: o_{e_k}, ACP: C_{e_k}. For this case, i. e., AIP: o_{e_k}, nobody except the organizer should learn the identities of the committed users, so the private row-keys have to be kept secret. And as not even invitees (who need to know CLL to be able to commit to the event) should learn the count of attendees, the organizer would need to add dummy entries on the commit-list to hide the count of attendees from the invitees. But this will also hide it from the attendees, as they do not have the private row-keys to tell apart dummy entries from normal entries, so ACP: C_{e_k} is not fulfilled.

A summary of how CLL and the private row-keys $rk_1^S \dots rk_n^S$ are shared depending on the settings for properties AIP and ACP is shown in Table 2.

Table 2. Sharing of CLL and $rk_1^S \dots rk_n^S$ as per the AIP and ACP settings. P = publicly available in $event_k$, I = privately shared with I_{e_k}, C = shared only with C_{e_k} (via the commit-disclose protocol), S = fully secret (only o_{e_k} knows about it).

Settings		Implementation			
AIP	**ACP**	$\boldsymbol{CLL}$	$\boldsymbol{rk_1^S \dots rk_n^S}$	**dummies**	**notes**
U	U	P	P	-	
I_{e_k}	U	P	I	-	
	I_{e_k}	P/I	I	if CLL public	
C_{e_k}	U	P	C	-	
	I_{e_k}	P/I	C	if CLL public	
	C_{e_k}	P	C	necessary	
o_{e_k}	U	P	S	-	
	I_{e_k}	I	S	-	
	C_{e_k}	-	-	-	not possible
	o_{e_k}	P	S	necessary	

4.6 Attendee-only Information Reliability Property

To implement this property, we will again use the commit-disclose protocol. The organizer o_{e_k} shares a private description $d_{e_k}^S$, encrypted with the key PDK, with the committed users C_{e_k}. The key is shared with these users in the disclose-list as soon as they store a valid commitment $c_{e_k}^{u_j}$ in the commit-list. The organizer o_{e_k} cannot have different private descriptions for groups of attendees of the same event e_k because they will all see the same ciphertext in the event object $event_k$. A cheating organizer o_{e_k} will be caught in the same manner as described above: if a user u_j commits and receives an invalid decryption key PDK, she

will publish the private disclose key dk_i^S to prove that she did not receive the promised private description $d_{e_k}^S$.

5 Discussion

The implementation presented realizes the event invitation feature in a decentralized system and fulfills the requirements of all of the defined security and privacy properties. Except for one parameter combination of the attendee identity/count privacy properties we were able to present implementation solutions for all possible choices of the tunable properties IIP, ICP, AIP and ACP.

An honest but curious user does not learn anything more than what is specified by the privacy settings.

A general limitation of our approach is, however, that for all properties based on the commit-disclose protocol, a malicious organizer is still able to cheat. But it disincentives her to do so as it provides a reliable cheating detection mechanism and offers the affected users the possibility to blame a cheating organizer – either publicly or in front of a chosen set of users, e. g., only other invitees of the event. We consider this an effective protection in the social scenarios that we see as possible application contexts of the event invitation feature. User identifiers are long-lived there and costly to change (as all friends have to be informed about a new identity), so we assume users care about their reputation and will try to avoid being exposed as misbehaving. Another limitation of our approach is the general problem of information usage control, i. e., insiders can always leak information to parties that should not learn this information according to a chosen privacy setting. For example, if only the invitees should learn the identities of other invited users, this can be violated by an invitee simply publishing the invite-list.

Some of the privacy protections are not secure against very powerful adversaries. For example the link obfuscation technique described in Section 4.2 relies on the unlinkability of the encrypted list object and the event object. This will be decreased by access patterns of invited users (if they are known), the structure/size of the list object (if distinguishable from other storage objects) and the entropy of the addressing scheme for storage objects. An adversary with the capability to pervasively monitor a large fraction of network traffic might be able to correlate requests for a certain event object and related list objects.

Finally, depending on the choice of privacy settings, the protocols not only allow the participants, i. e., organizer, invitees and attendees, to verify each others' claims, but also, to show the proof to an outsider. Such a process can be implemented in a client and used as one of the inputs for a reputation system, although this is out of the scope of this work.

6 Conclusion and Future Work

We have described and formalized a set of security and privacy properties for the event invitations feature in DOSNs, such as invitee/attendee identity privacy

(who learns the identities of the invitees/attendees), invitee/attendee count privacy (who learns the count of invitees/attendees), and attendee-only information reliability (availability of information exclusive to the attendees).

We described privacy enhancing tools, such as storage location indirection (to control not only who can decrypt an object but also who can see the ciphertext), controlled ciphertext inference (to allow a controlled information leak, e. g., about the size of an encrypted object to parties not able to decrypt the content) and a commit-disclose protocol to disclose a secret only to users who committed to attend an event and to detect a misbehaving party. Using these tools together with standard cryptographic primitives, we proposed a TTP-free architecture and decentralized protocols to implement the event invitation feature in a DOSN and analyzed the usability and privacy implications.

The results can be applied in the context of Privacy-Preserving DOSNs, but might also be useful in other domains such as Collaborative Working Environment and their corresponding collaborative-specific tools, i. e., groupware, for example, to perform tasks on shared documents. Another relevant domain is Massively Open Online Courses, for example, when restricting the access to lecture material of an online course to the registered students.

Possible future work includes evaluation of the performance, extending the security and privacy properties to include plausible deniability, anonymity or revocation, and extending the functionality of the feature to consider transferable invitation-rights or multiple organizers. Plausible deniability properties can be important when organizing political events. At the same time, it will probably introduce trade-offs with respect to the authenticity guarantees provided by the properties presented in this paper, e. g., the correctness of the attendee-count. Transferable invitation-rights would allow the organizer to specify a set of initially invited users, who then in turn can invite their friends to the event as well (but maybe limited to a certain number of hops in the social graph).

Acknowledgments

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

References

1. Baden, R., Bender, A., Spring, N., Bhattacharjee, B., Starin, D.: Persona: an online social network with user-defined privacy. In: Rodriguez, P., Biersack, E.W., Papagiannaki, K., Rizzo, L. (eds.) SIGCOMM. pp. 135–146. ACM (2009)
2. Buszko, D., Lee, W.H.D., Helal, A.: Decentralized ad-hoc groupware API and framework for mobile collaboration. In: GROUP. pp. 5–14. ACM (2001)
3. Cutillo, L.A., Molva, R., Strufe, T.: Safebook: A privacy-preserving online social network leveraging on real-life trust. IEEE Communications 47(12), 94–101 (2009)
4. El-Saddik, A., Rahman, A.S.M.M., Abdala, S., Solomon, B.: PECOLE: P2P multimedia collaborative environment. Multimedia Tools Appl. 39(3), 353–377 (2008)

5. Famulari, A., Hecker, A.: Mantle: A novel dosn leveraging free storage and local software. In: Guyot, V. (ed.) ICAIT. Lecture Notes in Computer Science, vol. 7593, pp. 213–224. Springer (2012)
6. Freitas, M.: twister - a P2P microblogging platform. CoRR abs/1312.7152 (2013)
7. Gilbert, H., Handschuh, H.: Security analysis of SHA-256 and sisters. In: Matsui, M., Zuccherato, R.J. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 3006, pp. 175–193. Springer (2003)
8. Greschbach, B., Kreitz, G., Buchegger, S.: The devil is in the metadata - new privacy challenges in decentralised online social networks. In: PerCom Workshops. pp. 333–339. IEEE (2012)
9. Greschbach, B., Kreitz, G., Buchegger, S.: User search with knowledge thresholds in decentralized online social networks. In: Hansen, M., Hoepman, J.H., Leenes, R.E., Whitehouse, D. (eds.) Privacy and Identity Management. IFIP Advances in ICT, vol. 421, pp. 188–202. Springer (2013)
10. Johnson-Lenz, P., Johnson-Lenz, T.: Groupware: Coining and defining it. SIGGROUP Bull. 19(2), 34– (Aug 1998)
11. Kim, M.K., Kim, H.C.: Awareness and privacy in groupware systems. In: CSCWD. pp. 984–988. IEEE (2006)
12. Li, W.D., Ong, S.K., Fuh, J.Y.H., Wong, Y.S., Lu, Y.Q., Nee, A.Y.C.: Feature-based design in a distributed and collaborative environment. Computer-Aided Design 36(9), 775–797 (2004)
13. Lunden, I.: Facebook turns off facial recognition in the EU, gets the all-clear on several points from Ireland's data protection commissioner on its review. Online (Sep 2012), http://techcrunch.com/2012/09/21/facebook-turns-off-facial-recognition-in-the-eu-gets-the-all-clear/
14. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2009), http://www.bitcoin.org/bitcoin.pdf
15. Reinhard, W., Schweitzer, J., Völksen, G., Weber, M.: CSCW tools: Concepts and architectures. IEEE Computer 27(5), 28–36 (1994)
16. Rodden, T., Blair, G.S.: CSCW and distributed systems: The problem of control. In: Bannon, L.J., Robinson, M., Schmidt, K. (eds.) ECSCW. Kluwer (1991)
17. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware. Lecture Notes in Computer Science, vol. 2218, pp. 329–350. Springer (2001)
18. Shih, G.: Facebook admits year-long data breach exposed 6 million users. Online (Jun 2013), http://www.reuters.com/article/2013/06/21/net-us-facebook-security-idUSBRE95K18Y20130621
19. Smith, C.: Reinventing social media: Deep learning, predictive marketing, and image recognition will change everything (Mar 2014), http://www.businessinsider.com/social-medias-big-data-future-2014-3
20. Trevor, J., Koch, T., Woetzel, G.: Metaweb: Bringing synchronous groupware to the world wide web. In: ECSCW. pp. 65–80 (1997)
21. Zhang, G., Jin, Q.: Scalable information sharing utilizing decentralized p2p networking integrated with centralized personal and group media tools. In: AINA (2). pp. 707–711. IEEE Computer Society (2006)
22. Zurko, M.E.: IBM Lotus Notes/Domino: Embedding Security in Collaborative Applications, chap. 30, pp. 607–622. O'Reilly Media, Inc. (2005)