Anonymously Sharing Flickr Pictures with Facebook Friends

Jan Camenisch Günter Karjoth Gregory Neven
Franz-Stefan Preiss

IBM Research – Zurich

{jca, nev, frp}@zurich.ibm.com, karjoth@acm.org

Abstract

Many Internet users today use an electronic social network service (SNS) to share data with their friends. Most SNSs let users restrict access to their shared data, e.g., to particular groups of friends, or to users satisfying other criteria based on their attributes or relationships. Usually, however, such access control restrictions can only be applied to resources hosted on the SNS itself. In this paper, we present protocols to enable SNS users to protect access to resources that are hosted on external service providers (SPs). Our mechanisms preserve the users' privacy in the sense that (1) the SP does not learn the SNS-identities of users that share or access the resource, nor does it learn anything about the access policy that protects it, (2) the SNS does not obtain any information about the resource, and in particular, does not obtain a link to it, and (3) the SP cannot change the policy set by the owner of the resource, or test the policy on users who never requested access to the resource. We give formal definitions of these security requirements and present a cryptographic protocol based on group signatures that provably fulfills them. We also discuss to what extent our requirements can be fulfilled using the standard OAuth authorization protocol while making only minor changes to the SNS infrastructure.

1 Introduction

Many individuals in today's society maintain profiles in one or more electronic social network services (SNSs) such as Facebook, LinkedIn, or Google+. Such SNSs allow their users to establish friend relationships with other users of the same SNS. Many SNSs also allow users to categorize their friends into groups or circles (e.g., best friends, work colleagues, family). These can then be used to restrict access to certain profile data or media (such as status messages and pictures) to members of a particular group. However, users cannot utilize their friend relationships and their categorizations that they established on one SNS to control access to their resources that are hosted on another SNS. Instead, they have to maintain the same relationships and groups on all SNSs they use. This is a major issue because the establishment and maintenance of friend relationships and groups is a continuous and very time-consuming process, and also entails a lock-in effect that becomes stronger the more friends a user maintains. For example, consider a user Alice who has a Facebook profile with numerous carefully selected friends as well as several well-maintained friend groups. When Alice wants to make use of another SNS, say Flickr, to share her pictures of the latest party with

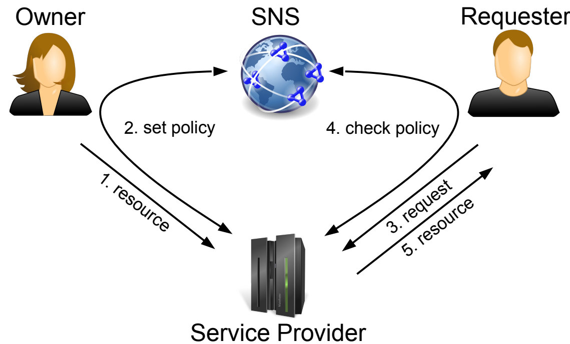

Figure 1: Interaction diagram

her Facebook *best friends* group (and only with them), all her best friends on Facebook need to join Flickr, Alice subsequently needs to accept them as friends and add them to her *best friends* group on Flickr so that they and only they can look at the picture.

In this paper we are concerned with schemes that spare Alice and her friends this tedious process and allow her to share her pictures securely and easily. In particular, we consider the scenario sketched in Figure 1. There are a number of users who are all registered with a social network service $\mathcal{SN}$, such as Facebook, where they have accounts. On $\mathcal{SN}$, users manage their friend relationships and groups. $\mathcal{SN}$ also knows other information about its users, which might be relevant to our scenario. A user $\mathcal{U}_i$ owns a certain resource (e.g., a collection of pictures) and wants to share it with her friends by depositing it (Step 1 in Figure 1) with a service provider $\mathcal{SP}$ such as Flickr. Because $\mathcal{U}_i$ doesn't want to share the resource with all of her friends, but only those who satisfy certain criteria, she creates an access policy plc describing who is allowed access to the resource and who is not. She then associates the access control policy (or a suitable encryption thereof) with the resource at $\mathcal{SP}$ or at $\mathcal{SN}$, depending on which of our two schemes is used (Step 2). As $\mathcal{U}_i$ manages her friends on $\mathcal{SN}$, the policy will be evaluated by $\mathcal{SN}$, but it will be enforced by $\mathcal{SP}$. We do not make assumptions about the format of the policy itself, but consider the details of the specification and the evaluation of the policy outside of the scope of this paper. To preserve privacy, we want that $\mathcal{SP}$ learns neither the policy nor the identities that $\mathcal{U}_i$ or her friends have with $\mathcal{SN}$. More precisely, if a user $\mathcal{U}_j$ requests access to the resource (Step 3), then $\mathcal{SP}$, $\mathcal{SN}$, and $\mathcal{U}_j$ interact to determine whether $\mathcal{U}_j$ should be granted access (Step 4). In this process, $\mathcal{SP}$ only learns whether or not $\mathcal{U}_j$ satisfies the policy plc as evaluated by $\mathcal{SN}$. In this basic scenario, we assume for simplicity of our exposition that users are known under their accounts with $\mathcal{SN}$; whether they also have accounts with $\mathcal{SP}$ is not relevant to our model. Our scheme can easily be extended to handle several resource hosting sites. How Alice informs her friends that she has made the resource available on $\mathcal{SP}$ is also out of scope and actually an orthogonal issue: she could use any method she prefers, such as sending an email, posting a link on her blog or on her wall on $\mathcal{SN}$, or perhaps her friends just regularly visit her page on $\mathcal{SP}$.

Ignoring the privacy requirements, one possible solution for the access control problem above is that Alice allows $\mathcal{SP}$ to query her $\mathcal{SN}$ profile data including her friend list. To this end, Alice could generate an OAuth token as to authorize $\mathcal{SP}$ to make the query on $\mathcal{SN}$. If $\mathcal{SN}$ is Facebook, this solution could indeed be implemented by a service provider $\mathcal{SP}$ already today using the APIs

that Facebook offers. A major drawback of such a naive solution, however, is that $\mathcal{SP}$ is able to make all kinds of queries to $\mathcal{SN}$. Even if the OAuth token is somehow bound to the access policy plc, $\mathcal{SP}$ will still learn the user names of Alice and her friends who accesses a resource. Thus, such a solution does not protect the privacy of Alice and her friends. Carminati and Ferrari pointed out that "relationships are in general sensitive resources whose privacy should be properly guaranteed even if they are instrumental to perform access control" [9].

Our Contributions We present the first solutions that enable SNS users to share their externally hosted resources with SNS friends while retaining a maximum level of privacy with respect to the service provider and the social network. To this end, we first present security definitions of a SNS-based access control scheme. Second, we provide a concrete realization based on group signatures [11], public key encryption with labels [23], commitment schemes, and one-time signature schemes [19] and prove our scheme to be secure, i.e., to meet our security definitions. Finally, we discuss how far one could meet our security definitions with implementations based on OAuth, in particular, what properties can be achieved while keeping the changes to existing systems as small as possible.

We believe that our schemes offer attractive features and argue that social networks would benefit from its implementation. In particular, they enable a centralization of friend relationship data and friend group data in a single SNS and reusing these relationships at other SNSs or external service providers for access control. This makes the maintenance of relationships much easier because the profile data must be maintained only in a single SNS rather than in multiple systems. Also, maintaining the profile data such as friend relationships and groups in the SNS ensures that access decisions that depend on this profile data are up to date. Further, a service provider that offers this kind of access control w.r.t. an external SNS makes its use more attractive as users no longer need to maintain new friend groups on it and does not require their friends to create their own accounts and profiles with the service provider as well.

The main feature of our scheme, however, is that it protects the privacy of the users. The service provider that controls access to the resources it hosts based on profile information maintained in a SNS does not learn the SNS identities of the profile owners or access requesters. To avoid the involved re-establishment of friend relationships and re-definition of friend groups in a newly joined SNS, users often resort to giving up their privacy by making their entire profile information publicly visible. As access to profile information can with our scheme easily be controlled based on relationships established in other SNSs, there is no longer the obstacle of re-establishing relationships and groups and thus no longer the need to resort to making all profile information public.

Of course, an SNS might want to internalize the functionality offered by possible competing service providers rather than implementing a scheme such as the one we propose. However, we believe that the latter is more attractive because many SNSs offer identity provider services to other sites already today.

Related Work There are many proposals to improve the access control systems of SNSs but focus has been on empowering users to control access within an SNS. Beyond marking data to be public, private, or accessible by direct contacts, access control systems deployed today provide different notions of personal relationships: friends, friends-of-friends, and circles for example. However, access control is restricted to the community of the Social Network; i.e., which members of the SNS can access what data stored within the SNS.

In this paper, we contribute to two of the protection requirements postulated by Gates [16] to handle the social networking capabilities provided by Web 2.0 technologies.

The first requirement states that access control must be *relationship-based*; i.e., the control of access to particular data should be based on the data owner's personal relationship to the recipient. Under the term Relationship-Based Access Control, a number of models have been developed. In [14] and subsequent work, Fong presents a model and language to express binary relations between the resource owner and the resource accessor. The model addresses the contextual nature of relationships and facilitates the composition of policies. Our protocols complement this work by providing privacy-preserving mechanisms to retrieve binary relationships at the time of policy evaluation.

Gates' second protection requirement states that there must be *interoperability* between the multiple sites; i.e., access control policies and relationship groups defined by the user should follow the user. To our knowledge, we are the first to propose a solution that provides an easy access to relationship data such that access control at the service provider can use relationship information maintained at the SNS in a privacy-friendly way.

Hu, Ahn, and Jorgensen proposed and implemented a solution for collaborative management of shared data [18]. Using multiparty policy evaluation techniques, data access is under supervision of the SNS controller.

Carminati and Ferrari pointed out that disclosing a relationship always means an exposure of personal information [9]. By hiding the identities of users to $\mathcal{SP}$, our protocols provide a privacy-aware access control mechanism, able to enforce relationship-based access control over multiple SNSs but ensuring relationship privacy.

Carminati et al. [10] make use of a semi-decentralized architecture, where the information concerning users' relationships is encoded into certificates, stored by a certificate server. The access control is enforced by the owner, who also plays the role of SP, and learns the identity of the requester in the process. In our approach, a selected SNS takes over the role of the certificate server, but the SP does not learn the identities of the involved users.

2 SNS-Based Access Control

The participants in an SNS-based access control system are a social network $\mathcal{SN}$ where users manage their profiles and friends lists, a number of users $\mathcal{U}_1, \ldots, \mathcal{U}_n$ who are registered with $\mathcal{SN}$, and a service provider $\mathcal{SP}$ which users use to host and share resources with other users. Users may or may not have accounts with $\mathcal{SP}$; since all access control decisions are taken based on the users' identities with $\mathcal{SN}$, this is irrelevant for our system. For simplicity, we consider i to be the user name of user $\mathcal{U}_i$ with $\mathcal{SN}$.

In the following, we first describe the high-level idea of our system, then we provide formal definitions of its algorithms, and finally we specify our security requirements.

2.1 High-Level Idea

To setup the system, $\mathcal{SN}$ runs a key generation protocol and publishes the system parameters of the scheme and $\mathcal{SN}$'s public key. To be able to share or to access a resource, a user $\mathcal{U}_i$ needs to first run a registration protocol with the social network $\mathcal{SN}$ from which the user obtains some registration information sk_i (the user's secret key).

Next, when a user $\mathcal{U}_i$ (owner of the resource) wants to deposit some resource with description res (this could for instance be the resource URL), she uses sk_i and res to generate what we call an *owner token* ot for some access control policy plc. $\mathcal{U}_i$ then deposits the owner token and the resource with $\mathcal{SP}$, who checks the validity of the token, and then she sends res by some adequate means to her friends (cf. Section 1).

Now, when a user $\mathcal{U}_j$ (requester of the resource) wants to access the resource with description res, he generates a *requester token* rt for the resource res using his sk_j and sends rt to $\mathcal{SP}$ as part of the request for the resource. $\mathcal{SP}$ uses ot and rt to generate what we call a linking token lt proving that the ot and rt relate to the same resource, and then sends the triple $(\mathsf{ot}, \mathsf{rt}, \mathsf{lt})$ to $\mathcal{SN}$. The linking token will ensure to $\mathcal{SN}$ that the requester token rt is indeed requesting access to the resource for which ot defines the policy in such a way that $\mathcal{SN}$ does not learn information about the resource. The generation of the linking token also includes a verification of the requester token.

Given these three tokens, $\mathcal{SN}$ can extract the policy plc and the user's identities $\mathcal{U}_i$ and $\mathcal{U}_j$. Thus, in some sense, by generating the requester and the owner tokens, $\mathcal{U}_i$ and $\mathcal{U}_j$ authorize $\mathcal{SP}$ to query $\mathcal{SN}$ for the evaluation of the policy plc w.r.t. their accounts with $\mathcal{SN}$. Next, $\mathcal{SN}$ evaluates whether or not $\mathcal{U}_j$ satisfies the policy plc w.r.t. $\mathcal{U}_i$ and sends the result (yes or no) back to $\mathcal{SP}$, who will allow or deny access to the resource based on the result.

To protect the privacy of users, we want that $\mathcal{SP}$ and $\mathcal{SN}$ do not learn more information than strictly necessary, i.e., $\mathcal{SN}$ should not learn any information about the resource, while $\mathcal{SP}$ doesn't learn the users' identities with $\mathcal{SN}$ nor any information about the policy. For security, we want that users who do not satisfy a policy cannot forge requester tokens that will give them access, and that the service provider $\mathcal{SP}$ cannot forge owner tokens to evaluate policies w.r.t. $\mathcal{U}_i$ that were not created by $\mathcal{U}_i$.

As mentioned already, we also want that owner tokens and requester tokens are somehow "linked" to the resource, so that they cannot be reused to protect or obtain access to a different resource. We want $\mathcal{SN}$ to be able to check that the owner token and requester token were created for the same resource, but at the same time we don't want $\mathcal{SN}$ to learn anything about res itself. We address these seemingly contradicting requirements by letting users send to $\mathcal{SP}$, along with each owner or requester token, also *linking information* oli or rli, respectively. The service provider doesn't forward oli and rli to $\mathcal{SN}$, but instead generates a *linking token* lt from oli and rli that essentially strips all information about res itself, but still allows $\mathcal{SN}$ to verify that ot and rt were indeed created for the same resource.

2.2 Definitions

An SNS-based access control system consists of five procedures: Setup, $\mathsf{Register}$, OTGen, OTVf, RTGen, LTGen, and $\mathsf{Extract}$. These procedures are defined as follows:

Setup A probabilistic algorithm to generate the keys for $\mathcal{SN}$. On input of a security parameter ℓ, the algorithm outputs the secret $\mathsf{sk}_{\mathcal{SN}}$ and public key $\mathsf{pk}_{\mathcal{SN}}$.

$\mathsf{Register}$ A probabilistic protocol between a user $\mathcal{U}_i$ and $\mathcal{SN}$. The user's input is $(i, \mathsf{pk}_{\mathcal{SN}})$ and $\mathcal{SN}$'s input is $(i, \mathsf{sk}_{\mathcal{SN}})$. The user's output is sk_i. We assume that the protocol will fail for an index i if it was already run for this index.

OTGen A probabilistic algorithm allowing a user $\mathcal{U}_i$ to generate an owner token ot together with linking information oli for resource $\mathsf{res} \in \{0,1\}^*$ stored by $\mathcal{SP}$ with access control policy $\mathsf{plc} \in \{0,1\}^*$ to be verified by $\mathcal{SN}$. They are computed as $(\mathsf{ot}, \mathsf{oli}) \leftarrow_R \mathsf{OTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_i, \mathsf{plc}, \mathsf{res})$.

OTVf A deterministic algorithm allowing $\mathcal{SP}$ to verify an owner token ot and its linking information oli w.r.t. res and $\mathcal{SN}$, i.e., $0/1 \leftarrow \mathsf{OTVf}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{oli}, \mathsf{res})$.

RTGen A probabilistic algorithm for allowing a user $\mathcal{U}_i$ to generate a requester token rt together with linking information rli for a resource res hosted by $\mathcal{SP}$ by computing $(\mathsf{rt}, \mathsf{rli}) \leftarrow_R \mathsf{RTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_i, \mathsf{res})$.

LTGen A possibly probabilistic algorithm that combines the policy linking information oli and access linking information rli into a linking token lt, i.e., $\mathsf{lt} \leftarrow_R \mathsf{LTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{oli}, \mathsf{rt}, \mathsf{rli}, \mathsf{res})$. It returns $\mathsf{lt} = \perp$ to indicate failure.

$\mathsf{Extract}$ A deterministic algorithm for allowing $\mathcal{SN}$ to extract the user identities and policies associated with a triple of a policy, an access, and a linking token, i.e., $(i, j, \mathsf{plc}) \leftarrow \mathsf{Extract}(\mathsf{sk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{rt}, \mathsf{lt})$. It returns $\perp$ if the tokens are malformed. $\mathcal{SN}$ can subsequently check whether users i and j satisfy the policy plc. Note that this algorithm does not require res as input. In fact, we will require that $\mathsf{ot}, \mathsf{rt}, \mathsf{lt}$ do not leak information about res (cf. Section 2.3).

2.3 Security Requirements

With respect to security and privacy, we require that our system fulfills the following properties:

Correctness We require that if all parties honestly run the protocols then the owner and requester tokens are valid and $\mathcal{SN}$ will be able to recover the policy and the identities of the users who generated the owner and requester tokens, respectively. That is, for all registered users $\mathcal{U}_i$ and $\mathcal{U}_j$ with respective secret keys sk_i and sk_j and for all $\mathsf{plc} \in \{0,1\}^*$, whenever $(\mathsf{ot}, \mathsf{oli}) \leftarrow_R \mathsf{OTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_i, \mathsf{plc}, \mathsf{res})$, $(\mathsf{rt}, \mathsf{rli}) \leftarrow_R \mathsf{RTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_j, \mathsf{res})$, and $\mathsf{lt} \leftarrow_R \mathsf{LTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{oli}, \mathsf{rt}, \mathsf{rli}, \mathsf{res})$, we require that $1 = \mathsf{OTVf}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{oli}, \mathsf{res})$, that $\mathsf{lt} \neq \perp$, and that $(i, j, \mathsf{plc}) = \mathsf{Extract}(\mathsf{sk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{rt}, \mathsf{lt})$ for all random choices of all the involved algorithms.

Anonymity We require that $\mathcal{SP}$ cannot tell which user identity registered with $\mathcal{SN}$ created an owner or requester token, or even whether two owner or requester tokens were generated by the same or by different users. We also require that the owner token leaks no information about the policy to $\mathcal{SP}$. More formally, we require that an adversary $\mathcal{A}$ (e.g., a malicious $\mathcal{SP}$) cannot win the following game with probability better than 1/2.

1. Run $(\mathsf{sk}_{\mathcal{SN}}, \mathsf{pk}_{\mathcal{SN}}) \leftarrow \mathsf{Setup}(\ell)$ and send $\mathsf{pk}_{\mathcal{SN}}$ to $\mathcal{A}$.
2. Run $\mathsf{sk}_1 \leftarrow \mathsf{Register}(1, \mathsf{pk}_{\mathcal{SN}})(1, \mathsf{sk}_{\mathcal{SN}})$ as well as $\mathsf{sk}_2 \leftarrow \mathsf{Register}(2, \mathsf{pk}_{\mathcal{SN}})(2, \mathsf{sk}_{\mathcal{SN}})$.
3. Allow $\mathcal{A}$ to repeatedly query any of the following oracles:
   (a) $\mathsf{Register}$ with $\mathcal{SN}$ as user $\mathcal{U}_i$ with $3 \leq i \leq n$.
   (b) Obtain an owner token ot with linking information oli from user $\mathcal{U}_1$ or $\mathcal{U}_2$ for some plc and res.
   (c) Obtain a requester token rt with linking information rli from user $\mathcal{U}_1$ or $\mathcal{U}_2$ for some res.
   (d) Obtain $(i, j, \mathsf{plc}) \leftarrow \mathsf{Extract}(\mathsf{sk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{rt}, \mathsf{lt})$ for any $\mathsf{ot}, \mathsf{rt}, \mathsf{lt}$.
4. Upon receiving two equal-length policies $\mathsf{plc}_1^*, \mathsf{plc}_2^*$ and resource res^* from $\mathcal{A}$, choose a random bit b and send $(\mathsf{ot}^*, \mathsf{oli}^*) \leftarrow_R \mathsf{OTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_{1+b}, \mathsf{plc}_{1+b}^*, \mathsf{res})$ and $(\mathsf{rt}^*, \mathsf{rli}^*) \leftarrow_R \mathsf{RTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_{2-b}, \mathsf{res})$ to $\mathcal{A}$.

5. Allow $\mathcal{A}$ to continue querying the above oracles, except that it cannot make queries to the $\mathsf{Extract}$ oracle involving $\mathsf{ot} = \mathsf{ot}^*$ or $\mathsf{rt} = \mathsf{rt}^*$.
6. Obtain a bit b' from $\mathcal{A}$, who wins if $b = b'$.

Resource Secrecy We require that the owner token, requester token, and linking token do not leak any information to $\mathcal{SN}$ about the resource res for which the tokens were generated. This is formalized as follows.

1. Run $(\mathsf{sk}_{\mathcal{SN}}, \mathsf{pk}_{\mathcal{SN}}) \leftarrow \mathsf{Setup}(\ell)$ and send $(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_{\mathcal{SN}})$ to $\mathcal{A}$.
2. Obtain two equal-length resources $\mathsf{res}_0, \mathsf{res}_1$ and a policy plc from the adversary $\mathcal{A}$.
3. Interact with $\mathcal{A}$ in two $\mathsf{Register}$ protocols to obtain secret keys sk_1 and sk_2 for users $\mathcal{U}_1$ and $\mathcal{U}_2$, respectively.
4. Choose a random bit b, compute $(\mathsf{ot}^*, \mathsf{oli}^*) \leftarrow_R \mathsf{OTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_1, \mathsf{plc}, \mathsf{res}_b)$, $(\mathsf{rt}^*, \mathsf{rli}^*) \leftarrow_R \mathsf{RTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{sk}_2, \mathsf{res}_b)$, and $\mathsf{lt}^* \leftarrow_R \mathsf{LTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{ot}^*, \mathsf{oli}^*, \mathsf{rt}^*, \mathsf{rli}^*, \mathsf{res}_b)$ and send $(\mathsf{ot}^*, \mathsf{rt}^*, \mathsf{lt}^*)$ to $\mathcal{A}$.
5. If $\mathcal{A}$ outputs $b' = b$ then it wins the game.

Token Unforgeability We require that a cheating user cannot get access if she does not satisfy the policy. At the same time, we want to prevent a cheating service provider $\mathcal{SP}$ from performing more policy evaluations than those that are strictly required to make its access decisions. This means that $\mathcal{SP}$ should not be able to modify the policy created by the resource owner, to evaluate policies on users that never requested access to any resource at all, nor to evaluate some policy on a user who never requested access to a resource to which this policy was associated. Technically, these restrictions all translate into the unforgeability of owner and requester tokens pointing to honest users when extracted through the $\mathsf{Extract}$ algorithm.

More formally, we require that no adversary $\mathcal{A}$ (e.g., a malicious $\mathcal{SP}$) can win the following game with non-negligible probability.

1. Run $(\mathsf{sk}_{\mathcal{SN}}, \mathsf{pk}_{\mathcal{SN}}) \leftarrow \mathsf{Setup}(\ell)$ and send $\mathsf{pk}_{\mathcal{SN}}$ to $\mathcal{A}$.
2. Run the protocols $\mathsf{sk}_1 \leftarrow \mathsf{Register}(1, \mathsf{pk}_{\mathcal{SN}})(1, \mathsf{sk}_{\mathcal{SN}})$ as well as $\mathsf{sk}_2 \leftarrow \mathsf{Register}(2, \mathsf{pk}_{\mathcal{SN}})(2, \mathsf{sk}_{\mathcal{SN}})$.
3. Allow $\mathcal{A}$ to repeatedly perform any of the steps below:
   (a) $\mathsf{Register}$ with $\mathcal{SN}$ as user $\mathcal{U}_i$ with $3 \leq i \leq n$.
   (b) Obtain an owner token ot with linking information oli from user $\mathcal{U}_1$ or $\mathcal{U}_2$ for some plc and res.
   (c) Obtain a requester token and linking information rli from user $\mathcal{U}_1$ or $\mathcal{U}_2$ for some res.
   (d) Obtain $(i, j, \mathsf{plc}) \leftarrow \mathsf{Extract}(\mathsf{sk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{rt}, \mathsf{lt})$ for any ot, rt, and lt.
4. When $\mathcal{A}$ eventually outputs $(\mathsf{ot}', \mathsf{rt}', \mathsf{lt}')$, check that $(i, j, \mathsf{plc}') \leftarrow \mathsf{Extract}(\mathsf{sk}_{\mathcal{SN}}, \mathsf{ot}', \mathsf{rt}', \mathsf{lt}')$ doesn't return $\perp$. $\mathcal{A}$ wins the game if at least one of the following conditions is satisfied:
   (a) $i = 1$ and $\mathcal{A}$ never queried an owner token from $\mathcal{U}_1$ for policy plc'.
   (b) $j = 2$ and $\mathcal{A}$ never queried a requester token from $\mathcal{U}_2$.
   (c) $i = 1$, $j = 2$, and there does not exist a resource res' such that $\mathcal{A}$ queried an owner token from $\mathcal{U}_1$ for plc' and res' and $\mathcal{A}$ queried a requester token for $\mathcal{U}_2$ for res'.

Provider Security We want to ensure that no adversary is able to produce tokens that pass the owner token verification algorithm and that $\mathcal{SP}$ can use to create a valid linking token, but for which $\mathsf{Extract}$ fails. More formally, we require that no adversary $\mathcal{A}$ can win the following game with non-negligible probability.

1. Run $(\mathsf{sk}_{\mathcal{SN}}, \mathsf{pk}_{\mathcal{SN}}) \leftarrow \mathsf{Setup}(\ell)$ and send $\mathsf{pk}_{\mathcal{SN}}$ to $\mathcal{A}$.
2. Allow $\mathcal{A}$ to repeatedly perform any of the steps below:
   (a) $\mathsf{Register}$ with $\mathcal{SN}$ as user $\mathcal{U}_i$ with $1 \leq i \leq n$.
   (b) Obtain $(i, j, \mathsf{plc}) \leftarrow \mathsf{Extract}(\mathsf{sk}_{\mathcal{SN}}, \mathsf{ot}, \mathsf{rt}, \mathsf{lt})$ for any $\mathsf{ot}, \mathsf{rt}, \mathsf{lt}$.
3. When $\mathcal{A}$ outputs $(\mathsf{ot}', \mathsf{oli}', \mathsf{rt}', \mathsf{rli}', \mathsf{res}'$, it wins the game if $1 = \mathsf{OTVf}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{ot}', \mathsf{oli}', \mathsf{res}')$, if $\mathsf{lt}' \neq \perp$ for $\mathsf{lt}' \leftarrow_R \mathsf{LTGen}(\mathsf{pk}_{\mathcal{SN}}, \mathsf{ot}', \mathsf{oli}', \mathsf{rt}', \mathsf{rli}', \mathsf{res}')$, and if $\perp = \mathsf{Extract}(\mathsf{sk}_{\mathcal{SN}}, \mathsf{ot}', \mathsf{rt}', \mathsf{lt}')$.

3 Group Signature Construction

Our main construction of an SNS-based access control scheme is based on group signatures, which allow a group of signers to anonymously sign messages in the name of the group such that only a group manager can figure out the exact signer. The role of the group manager is played by $\mathcal{SN}$, who gives out a signing key to each user that registers at $\mathcal{SN}$. The owner of a resource $\mathcal{U}_i$ attaches an access policy to the resource by encrypting the policy under $\mathcal{SN}$'s public key, creating a commitment to an identifier of the resource res, and computing a group signature on the ciphertext and the commitment. To make sure that the ciphertext cannot be separated from the owner token and used in combination with another group signature, we use known techniques involving one-time signatures [20, 8] to bind the ciphertext to the token. The owner token contains the group signature, the ciphertext, the commitment, a one-time signature on all three of these, and the public key to the one-time signature scheme; the linking information consists of the opening information to the commitment. When user $\mathcal{U}_j$ requests access to a resource, he also commits to the resource and signs the commitment using his group signing key. The service provider $\mathcal{SP}$ verifies all signatures in the tokens and checks that both commitments open to the resource identifier res. It then creates the linking token by computing a non-interactive zero-knowledge proof [5] that the commitments in the owner token and in the requester token open to the same resource. $\mathcal{SN}$ first verifies all signatures and the zero-knowledge proof, extracts the user identities i, j by opening the group signatures, and decrypts the access policy. It can then evaluate whether $\mathcal{U}_j$ satisfies the policy with respect to $\mathcal{U}_i$ and send the access decision back to $\mathcal{SP}$.

We now discuss the ingredients to our scheme and then proceed to present our main construction.

3.1 Preliminaries

3.1.1 Group Signatures

Group signatures [11] allow a signer to anonymously sign messages in name of a whole group of users, who obtain their secret signing keys from a group manager. The group manager is responsible for the initialization of the group and for the admission of group members. Anyone can verify that the message was signed by a valid group member using the group's public key, but only the group manager (or a dedicated opener) can recover the exact identity of the signer.

Depending on whether the signing keys are pre-computed during the initialization phase for all users, or dynamically generated whenever a new user joins the group, a group signature scheme is

called *static* [1] or *dynamic* [3]. For simplicity, we describe our construction in terms of static group signatures, but our constructions and results easily extend to the dynamic case.

A group signature scheme consists of the following algorithms:

GKg A probabilistic algorithm that the group manager uses on input of a security parameter ℓ and the number of users n to generate the group public key gpk, signing keys $\mathsf{gsk}_1, \ldots, \mathsf{gsk}_n$, and the opening key ok. It publishes gpk, hands signing key gsk_i to user $\mathcal{U}_i$, and keeps ok secret.

GSign A signing algorithm that user $\mathcal{U}_i$ runs on input of a signing key gsk_i and a message msg to create a group signature σ.

$\mathsf{GVerify}$ A deterministic verification algorithm that the verifier runs on input of the group public key gpk, a message msg and signature σ and returns one if the signature is valid and returns zero otherwise.

GOpen An opening algorithm that on input of the opening key ok, a message msg and a signature σ, returns the index $i \in \{1, \ldots, n\}$ of the user who created the signature or $\perp$ to indicate failure.

Correctness requires that for all $\ell, n, i \in \mathbb{N}$ and for all $\mathsf{msg} \in \{0,1\}^*$ we have $\mathsf{GVerify}(\mathsf{gpk}, \mathsf{msg}, \mathsf{GSign}(\mathsf{gsk}_i, \mathsf{msg})) = 1$ and $\mathsf{GOpen}(\mathsf{ok}, \mathsf{msg}, \mathsf{GSign}(\mathsf{gsk}_i, \mathsf{msg})) = i$ with probability one when $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_n, \mathsf{ok}) \leftarrow \mathsf{GKg}(\ell, n)$.

Security of group signature schemes consists of two properties, namely anonymity and traceability. Anonymity essentially guarantees that nobody except the group manager can tell which signer created a particular signature. More formally, it requires that no adversary $\mathcal{A}$ can win the following game with probability non-negligibly (in the security parameter ℓ) higher than $1/2$:

1. Run $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_n, \mathsf{ok}) \leftarrow \mathsf{GKg}(\ell, n)$ and give gpk as well as all signing keys $\mathsf{gsk}_1, \ldots, \mathsf{gsk}_n$ to $\mathcal{A}$.
2. Allow $\mathcal{A}$ to make repeated queries to an opening oracle $\mathsf{GOpen}(\mathsf{ok}, \cdot, \cdot)$.
3. When $\mathcal{A}$ outputs a message msg and user indices $i_0, i_1 \in \{1, \ldots, n\}$, choose a random bit $b \leftarrow \{0,1\}$, compute $\sigma \leftarrow \mathsf{GSign}(\mathsf{gsk}_i, \mathsf{msg})$, give σ to $\mathcal{A}$ and continue to run $\mathcal{A}$ with access to the opening oracle.
4. $\mathcal{A}$ wins the game if it outputs a bit $b' = b$ and it never queried the opening oracle on msg, σ.

Traceability guarantees that a valid group signature can always be traced back to a user whose signing key was used in the creation of the group signature. This implies unforgeability, since the ability to create a valid signature without access to any signing keys would violate traceability. This property is more formally defined through the following game:

1. Run $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_n, \mathsf{ok}) \leftarrow \mathsf{GKg}(\ell, n)$ and give gpk as well as ok to $\mathcal{A}$.
2. Allow $\mathcal{A}$ to repeatedly query a key oracle that on input of $i \in \{1, \ldots, n\}$ returns gsk_i and a signing oracle $\mathsf{GSign}(\mathsf{gsk}_\cdot, \cdot)$.
3. $\mathcal{A}$ wins the game if it outputs a message msg and signature σ so that $\mathsf{GVerify}(\mathsf{gpk}, \mathsf{msg}, \sigma) = 1$ and either $\mathsf{GOpen}(\mathsf{ok}, \mathsf{msg}, \sigma) = \perp$, or $\mathsf{GOpen}(\mathsf{ok}, \mathsf{msg}, \sigma) = i$ so that $\mathcal{A}$ never queried the key oracle on i nor the signing oracle on i, msg.

3.1.2 Public-Key Encryption with Labels

Public-key encryption with labels [23] is similar to standard public-key encryption, but one can bind a *label* to a ciphertext in a non-malleable way. Labels can be added to any standard public-key

encryption scheme by appending the label to the message, but many schemes offer more efficient instantiations. A scheme consists of a key generation algorithm EKg that on input of a security parameter ℓ generates a public encryption key epk and corresponding secret decryption key esk; an encryption algorithm Enc that on input of epk, a message msg and a label λ outputs a ciphertext c; and a decryption algorithm Dec that on input of esk, λ and c outputs msg or $\perp$ to indicate failure. Correctness requires that $\mathsf{Dec}(\mathsf{esk}, \lambda, \mathsf{Enc}(\mathsf{epk}, \mathsf{msg}, \lambda)) = \mathsf{msg}$ with probability one for any key pair $(\mathsf{epk}, \mathsf{esk}) \leftarrow \mathsf{EKg}(\ell)$, any $\ell \in \mathbb{N}$ and any $\mathsf{msg}, \lambda \in \{0,1\}^*$.

We use the standard security notion of indistinguishability under adaptive chosen-ciphertext attack (IND-CCA2) where the adversary $\mathcal{A}$ is given the encryption key epk and adaptive access to a decryption oracle $\mathsf{Dec}(\mathsf{esk}, \cdot, \cdot)$. When $\mathcal{A}$ outputs a label λ^* and two equal-length messages $\mathsf{msg}_0^*, \mathsf{msg}_1^*$, it is given $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg}_b^*, \lambda^*)$ for a random bit $b \in \{0,1\}$. The adversary wins the game if he outputs $b' = b$ without querying (λ^*, c^*) to the decryption oracle.

3.1.3 One-Time Signature Schemes

A one-time signature scheme consists of three algorithms $(\mathsf{OTKg}, \mathsf{OTSign}, \mathsf{OTVerify})$ where OTKg, on input of a security parameter ℓ, generates a public verification key otpk and a corresponding secret signing key otsk; OTSign takes as input otsk and a message msg to produce a signature ots; and $\mathsf{OTVerify}$ outputs a bit indicating whether a given signature ots is valid with respect to public key otpk and message msg. Correctness requires that $\mathsf{OTVerify}(\mathsf{otpk}, \mathsf{msg}, \mathsf{OTSign}(\mathsf{otsk}, \mathsf{msg})) = 1$ with probability one whenever $(\mathsf{otpk}, \mathsf{otsk}) \leftarrow_R \mathsf{OTKg}(\ell)$ for all $\ell \in \mathbb{N}$ and $\mathsf{msg} \in \{0,1\}^*$. The scheme is said to be *strongly one-time unforgeable* if no adversary running in time polynomial in ℓ can, on input of otpk and after a single query msg to a signing oracle $\mathsf{OTSign}(\mathsf{otsk}, \cdot)$ that returns ots, produce a forgery $(\mathsf{msg}', \mathsf{ots}')$ such that $\mathsf{OTVerify}(\mathsf{otpk}, \mathsf{msg}', \mathsf{ots}') = 1$ and $(\mathsf{msg}', \sigma') \neq (\mathsf{msg}, \mathsf{ots})$. Any (multiple-time) signature scheme that is strongly unforgeable under chosen-message attack is also a one-time strongly unforgeable scheme, but more efficient constructions exist [19].

3.1.4 Commitments with Equality Proofs

A commitment scheme consists of a parameter generation algorithm CPGen that, on input of security parameter ℓ, returns parameters cpars; a committing algorithm Commit that, on input of cpars and a message msg, returns a commitment cmt together with opening op; and an opening algorithm COpen that, on input of cpars, cmt, msg, and op, outputs a bit indicating whether cmt is a valid commitment to msg.

Security of commitment schemes consists of the *hiding* and the *binding* properties. It is hiding if no adversary $\mathcal{A}$ can, with non-negligible probability, on input of cpars generate two messages $\mathsf{msg}_0, \mathsf{msg}_1$, receive a commitment cmt^* of msg_b for a random bit b, and output $b' = b$. The binding property requires that no adversary $\mathcal{A}$ can output a commitment that opens correctly to two different messages, i.e., on input of cpars, output a commitment cmt, two messages $\mathsf{msg}_0 \neq \mathsf{msg}_1$, and opening information $\mathsf{op}_0, \mathsf{op}_1$ such that $\mathsf{COpen}(\mathsf{cmt}, \mathsf{msg}_0, \mathsf{op}_0) = \mathsf{COpen}(\mathsf{cmt}, \mathsf{msg}_1, \mathsf{op}_1) = 1$.

We also need a non-interactive zero-knowledge proof of knowledge [5] (NIZK PoK) that two commitments $\mathsf{cmt}_1, \mathsf{cmt}_2$ open to the same message, i.e., $\pi \leftarrow_R \mathsf{NIZK}\{(\mathsf{msg}, \mathsf{op}_1, \mathsf{op}_2) : \mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}_1, \mathsf{msg}, \mathsf{op}_1) = \mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}_2, \mathsf{msg}, \mathsf{op}_2) = 1\}$. The proof can be verified using $\mathsf{cpars}, \mathsf{cmt}_1, \mathsf{cmt}_2$. The proof system must be (1) complete, meaning that the verification algorithm accepts any honestly generated proof, (2) zero knowledge, meaning that there exists a zero-knowledge simulator that produces a valid proof for any two commitments $\mathsf{cmt}_1, \mathsf{cmt}_2$ without knowing the witnesses $\mathsf{msg}, \mathsf{op}_1, \mathsf{op}_1$, and (3) a proof of knowledge, meaning that for any adversary $\mathcal{A}$ producing a proof

π for $\mathsf{cmt}_1, \mathsf{cmt}_2$, there exists an extractor that, given black-box access to $\mathcal{A}$, recovers $\mathsf{msg}, \mathsf{op}_1, \mathsf{op}_2$ such that $\mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}_1, \mathsf{msg}, \mathsf{op}_1) = \mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}_2, \mathsf{msg}, \mathsf{op}_2) = 1$.

A well-known commitment scheme is due to Pedersen [21]. The parameters contain the description of a cyclic group $\mathbb{G}$ of prime order p and two random generators g, h. To commit to a message msg, one chooses $\mathsf{op} \leftarrow_R \mathbb{Z}_p$ and computes $\mathsf{cmt} = g^{\mathsf{msg}} h^{\mathsf{op}}$. To verify the opening of a commitment, one simply recomputes cmt from $\mathsf{msg}, \mathsf{op}$. The Pedersen scheme is unconditionally hiding and is binding under the discrete logarithm assumption in $\mathbb{G}$. One can obtain a NIZK proof of knowledge $\mathsf{NIZK}\{(\mathsf{msg}, \mathsf{op}_1, \mathsf{op}_2) \ : \ \mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}_1, \mathsf{res}, \mathsf{op}_1) = 1 \ \wedge \ \mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}_2, \mathsf{res}, \mathsf{op}_2) = 1\}$ using generalized Schnorr proofs [22, 7] through a Fiat-Shamir transformation [13] as follows.

1. The prover, on input of $\mathsf{msg}, \mathsf{op}_1, \mathsf{op}_2$, chooses random $r_0, r_1, r_2 \leftarrow_R \mathbb{Z}_p$ and computes $t_1 \leftarrow g^{r_0} h^{r_1}$ and $t_2 \leftarrow g^{r_0} h^{r_2}$.
2. It applies a hash function H to compute $c = \mathsf{H}(g, h, \mathsf{cmt}_1, \mathsf{cmt}_2, t_1, t_2)$.
3. It computes $s_0 \leftarrow c \cdot \mathsf{msg} + r_0 \bmod p$ and $s_i \leftarrow c \cdot \mathsf{op}_i + r_i \bmod p$ for $i = 1, 2$.
4. It returns $\pi = (c, s_0, s_1, s_2)$.

To verify the proof, one computes $t_i \leftarrow g^{s_0} h^{s_i} \bmod p$ for $i = 1, 2$ and checks that $c = \mathsf{H}(g, h, \mathsf{cmt}_1, \mathsf{cmt}_2, t_1, t_2)$. The proof is a NIZK PoK under the discrete logarithm assumption in $\mathbb{G}$ if H is modeled as a random oracle [2].

3.2 Construction

We build an SNS-based access control system from a group signature scheme ($\mathsf{GKg}, \mathsf{GSign}, \mathsf{GVerify}, \mathsf{GOpen}$), a public-key encryption scheme ($\mathsf{EKg}, \mathsf{Enc}, \mathsf{Dec}$), and a commitment scheme ($\mathsf{CPGen}, \mathsf{Commit}, \mathsf{COpen}$) with proof system NIZK as follows:

Setup On input of ℓ, the ESN $\mathcal{SN}$ runs $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_n, \mathsf{ok}) \leftarrow_R \mathsf{GKg}(\ell, n)$ where n is an upper bound for the number of users in the ESN. It also generates an encryption key pair $(\mathsf{epk}, \mathsf{esk}) \leftarrow_R \mathsf{EKg}(\ell)$ and commitment parameters $\mathsf{cpars} \leftarrow_R \mathsf{CPGen}(\ell)$. Return $\mathsf{sk}_{\mathcal{SN}} = (\mathsf{gsk}_1, \ldots, \mathsf{gsk}_n, \mathsf{ok}, \mathsf{esk})$ and $\mathsf{pk}_{\mathcal{SN}} = (\mathsf{gpk}, \mathsf{epk}, \mathsf{cpars})$.

Register When user $\mathcal{U}_i$ registers with an ESN $\mathcal{SN}$, the latter simply sends $\mathsf{sk}_i = \mathsf{gsk}_i$ to $\mathcal{U}_i$.

OTGen User $\mathcal{U}_i$ generates a owner token ot for a resource res hosted on $\mathcal{SP}$ with access control policy plc by generating a one-time signing key pair $(\mathsf{otpk}, \mathsf{otsk}) \leftarrow_R \mathsf{OTKg}(\ell)$, encrypting the policy under $\mathcal{SN}$'s public key with label otpk as $c \leftarrow \mathsf{Enc}(\mathsf{epk}, \mathsf{plc}, \mathsf{otpk})$, committing to the resource $(\mathsf{cmt}, \mathsf{op}) \leftarrow_R \mathsf{Commit}(\mathsf{cpars}, \mathsf{res})$, creating a group signature $\sigma \leftarrow \mathsf{GSign}(\mathsf{gsk}_i, (\mathtt{policy}, c, \mathsf{cmt}, \mathsf{otpk}, \mathcal{SP}))$, and signing $\mathsf{ots} \leftarrow_R \mathsf{OTSign}(\mathsf{otsk}, (c, \mathsf{cmt}, \sigma))$. He sends the owner token $\mathsf{ot} = (c, \mathsf{cmt}, \sigma, \mathsf{otpk}, \mathsf{ots})$ and linking information $\mathsf{oli} = \mathsf{op}$ to $\mathcal{SP}$.

OTVf The service provider $\mathcal{SP}$ verifies $\mathsf{ot} = (c, \mathsf{cmt}, \sigma, \mathsf{otpk}, \mathsf{ots})$ with $\mathsf{oli} = \mathsf{op}$ for resource res by checking that $\mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}, \mathsf{res}, \mathsf{op}) = 1$, that $\mathsf{GVerify}(\mathsf{gpk}, (\mathtt{policy}, c, \mathsf{cmt}, \mathsf{otpk}, \mathcal{SP}), \sigma) = 1$, and that $\mathsf{OTVerify}(\mathsf{otpk}, (c, \mathsf{cmt}, \sigma), \mathsf{ots}) = 1$.

RTGen When a user $\mathcal{U}_j$ wants to access a resource res hosted at $\mathcal{SP}$ it creates a commitment $(\mathsf{cmt}, \mathsf{op}) \leftarrow_R \mathsf{Commit}(\mathsf{cpars}, \mathsf{res})$, computes a group signature $\sigma \leftarrow \mathsf{GSign}(\mathsf{gsk}_j, (\mathtt{access}, \mathsf{cmt}, \mathcal{SP}))$, and sends requester token $\mathsf{rt} = (\mathsf{cmt}, \sigma)$ and linking information $\mathsf{rli} = \mathsf{op}$ to $\mathcal{SP}$.

LTGen The service provider $\mathcal{SP}$ computes the linking token lt as a NIZK PoK that the commitments in the policy and requester token are for the same resource. More precisely, it first verifies requester

token $\mathsf{rt} = (\mathsf{cmt}_2, \sigma_2)$ with $\mathsf{rli} = \mathsf{op}_2$ by checking that $\mathsf{COpen}(\mathsf{cpars}, \mathsf{res}, \mathsf{cmt}_2, \mathsf{op}_2) = 1$ and by verifying the signature $\mathsf{GVerify}(\mathsf{gpk}, (\mathtt{access}, \mathsf{cmt}_2, \mathcal{SP})) = 1$. If any of these tests fail, it returns $\perp$, otherwise it uses owner token $\mathsf{ot} = (c, \mathsf{cmt}_1, \sigma_1, \mathsf{otpk}, \mathsf{ots})$ and $\mathsf{oli} = \mathsf{op}_1$ to produce $\mathsf{lt} \leftarrow_R \mathsf{NIZK}\{(\mathsf{res}, \mathsf{op}_1, \mathsf{op}_2) : \mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}_1, \mathsf{res}, \mathsf{op}_1) = 1 \wedge \mathsf{COpen}(\mathsf{cpars}, \mathsf{cmt}_2, \mathsf{res}, \mathsf{op}_2) = 1\}$.

Extract When the ESN $\mathcal{SN}$ receives a owner token $\mathsf{ot} = (c, \mathsf{cmt}_1, \sigma_1, \mathsf{otpk}, \mathsf{ots})$, an requester token $\mathsf{rt} = (\mathsf{cmt}_2, \sigma_2)$, and a linking token lt from resource host $\mathcal{SP}$, it proceeds as follows. If any of the signatures is invalid, i.e., if $\mathsf{GVerify}(\mathsf{gpk}, (\mathtt{policy}, c, \mathsf{cmt}_1, \mathsf{otpk}, \mathcal{SP}), \sigma_1) = 0$, $\mathsf{OTVerify}(\mathsf{otpk}, (c, \mathsf{cmt}_1, \sigma_1), \mathsf{ots}) = 0$, or $\mathsf{GVerify}(\mathsf{gpk}, (\mathtt{access}, \mathsf{cmt}_2, \mathcal{SP}), \sigma_2) = 0$, or if verification of the NIZK PoK lt fails w.r.t. $\mathsf{cmt}_1, \mathsf{cmt}_2$, then it returns $\perp$. Otherwise, it opens σ_1 and σ_2 as $i \leftarrow \mathsf{GOpen}(\mathsf{ok}, (\mathtt{policy}, c, \mathsf{cmt}_1, \mathcal{SP}), \sigma_1)$ and $j \leftarrow \mathsf{GOpen}(\mathsf{ok}, (\mathtt{access}, \mathsf{cmt}_2, \mathcal{SP}), \sigma_2)$. It also decrypts the policy $\mathsf{plc} \leftarrow \mathsf{Dec}(\mathsf{esk}, c, \mathsf{otpk})$. If $i = \perp$ or $j = \perp$ then it returns $\perp$, otherwise it returns (i, j, plc), allowing $\mathcal{SN}$ to check whether i and j satisfy the policy plc.

3.3 Security Proof

Theorem 1. *If the one-time signature scheme is strongly one-time unforgeable, the public-key encryption scheme is IND-CCA2, and the group signature scheme is anonymous and traceable, then the preceding construction is anonymous.*

Sketch. We prove the above theorem through a sequence of games [24] Game 0 through Game 4. Game 0 is the anonymity game of the SNS-based access control scheme, while we show that any adversary winning Game 4 gives rise to an anonymity adversary for the underlying group signature scheme. By proving for each game hop that, under appropriate assumptions, no polynomial-time adversary can distinguish one game from the next with non-negligible probability, the overall scheme is proved secure. We briefly sketch the different games and the reductions from the assumptions underlying each hop.

Game 1 Identical to Game 0, except that in the second phase, whenever the Extract oracle is queried for a owner token $\mathsf{ot} = (c, \mathsf{cmt}, \sigma, \mathsf{otpk}, \mathsf{ots})$ containing the same one-time public key $\mathsf{otpk} = \mathsf{otpk}^*$ as the target owner token $\mathsf{ot}^* = (c^*, \mathsf{cmt}^*, \sigma^*, \mathsf{otpk}^*, \mathsf{ots}^*)$, it returns $\perp$. Any adversary distinguishing Game 0 from Game 1 gives rise to an attack on the strong unforgeability of the one-time signature scheme, because if $\mathsf{ot} = \mathsf{ot}^*$ and $\mathsf{otpk} = \mathsf{otpk}^*$, then either $(c, \mathsf{cmt}, \sigma) \neq (c^*, \mathsf{cmt}^*, \sigma^*)$ so that ots is a signature on a different message $(c, \mathsf{cmt}, \sigma)$, or $\mathsf{ots} \neq \mathsf{ots}^*$ so that ots is a different signature of the same message. Both of these cases break the strong one-time unforgeability of the scheme.

Game 2 Identical to Game 1, except that whenever the Extract oracle is queried for a owner token with $\mathsf{otpk} \neq \mathsf{otpk}^*$ and $\sigma = \sigma^*$, then it returns $\perp$. Any adversary that can distinguish between Game 0 and Game 1 must come up with a public key otpk such that the honestly generated group signature σ^* on otpk^* is also valid for otpk, which constitutes a forged group signature and therefore breaks the traceability property of the group signature scheme.

Game 3 Identical to Game 2, except that whenever the adversary queries the Extract oracle with an requester token $\mathsf{rt} = (\mathsf{cmt}, \sigma)$ with $\sigma = \sigma^*$ and $\mathsf{cmt} \neq \mathsf{cmt}^*$, where $\mathsf{rt}^* = (\mathsf{cmt}^*, \sigma^*)$ is the challenge requester token, then the oracle returns $\perp$. Any adversary distinguishing Game 2 from Game 3 must come up with a commitment $\mathsf{cmt} \neq \mathsf{cmt}^*$ such that σ^* is valid for both cmt and cmt^*, which constitutes a forgery of the group signature scheme and therefore contradicts traceability.

Game 4 Identical to Game 3, but the ciphertext in the challenge owner token is created as an encryption of ones, i.e., $c^* \leftarrow \mathsf{Enc}(\mathsf{epk}, 1^{|\mathsf{plc}_1|}, \mathsf{otpk}^*)$. Any adversary distinguishing between Games 3 and 4 can be used to break the IND-CCA2 property of the encryption scheme. In the reduction, the IND-CCA2 adversary $\mathcal{B}$ will use its decryption oracle to answer $\mathcal{A}$'s $\mathsf{Extract}$ queries. Note that $\mathcal{B}$ will never have to make the "forbidden" decryption query for c^* with label otpk^*, because all $\mathsf{Extract}$ queries involving otpk^* are answered with $\perp$ since Game 1.

Note that in Game 4, the ciphertext c^* in the challenge owner token is independent of the hidden bit b, and that all $\mathcal{A}$'s queries to the $\mathsf{Extract}$ oracle involving a group signature σ that is either taken from ot^* or from rt^* will be responded with $\perp$. Any adversary $\mathcal{A}$ winning Game 4 can therefore be used to build an adversary $\mathcal{B}$ against the anonymity of the group signature, where $\mathcal{B}$ uses gsk_i to respond to $\mathcal{A}$'s $\mathsf{Register}$, OTGen and RTGen queries and uses its GOpen oracle to respond to $\mathcal{A}$'s $\mathsf{Extract}$ queries. $\square$

Theorem 2. *If the commitment equality proof is zero knowledge and the commitment scheme is hiding, then our construction satisfies the resource secrecy property.*

Sketch. We prove the security through a sequence of games Game 0 through Game 2, where Game 0 is the original resource secrecy game.

Game 1 Identical to Game 0, but the game uses the zero-knowledge simulator to produce the NIZK PoK lt^*, so that it no longer needs witnesses $\mathsf{res}_b, \mathsf{op}_1, \mathsf{op}_2$ to produce it.

Game 2 Identical to Game 1, but the game commits to a string of ones instead of the real resource res_b, i.e., in the creation of the challenge policy and requester token it computes $(\mathsf{cmt}_i, \mathsf{op}_i) \leftarrow_R \mathsf{Commit}(\mathsf{cpars}, 1^{|\mathsf{res}_0|})$ for $i = 1, 2$. This game is indistinguishable from the previous one by the hiding property of the commitment scheme.

In the final game Game 2, the adversary's view is independent of the bit b chosen by the game, so its advantage in winning the game is 1/2. $\square$

Theorem 3. *If the group signature scheme is traceable, the commitment equality proof is a proof of knowledge, and the commitment scheme is binding, then our construction satisfies the token unforgeability property.*

Sketch. Let's distinguish between a Type-a, Type-b, and Type-c adversary depending on the different winning conditions that they satisfy in Step 4 of the winning game.

A Type-a adversary $\mathcal{A}$ outputs $(\mathsf{ot}', \mathsf{rt}', \mathsf{lt}')$ such that $(i, j, \mathsf{plc}') \leftarrow \mathsf{Extract}(\mathsf{sk}_{\mathcal{SN}}, \mathsf{ot}', \mathsf{rt}', \mathsf{lt}')$ with $i = 1$ and $\mathcal{A}$ never queried a owner token from $\mathcal{U}_1$ for policy plc', while for a Type-b adversary $j = 2$ and $\mathcal{A}$ never queried any requester token from $\mathcal{U}_2$. Both Type-a and Type-b adversaries are easily converted into forgeries for the group signature scheme, i.e., they break the traceability property of the group signature scheme.

The forger of a Type-c adversary is such that $i = 1$, $j = 2$, and there does not exist a resource res' such that $\mathcal{A}$ queried a owner token by $\mathcal{U}_1$ for $\mathsf{plc}', \mathsf{res}'$ as well as an requester token by $\mathcal{U}_2$ for res'. Here, we can distinguish between two adversary subtypes: a Type-c1 adversary, where one of ot' or rt' contain a commitment cmt_1 or cmt_2 that was not previously returned as part of a policy or requester token by the OTGen or RTGen oracle, respectively, and a Type-c2 adversary, where both commitments were recycled from previous tokens. A Type-c1 adversary again easily gives rise to a forgery on the group signature scheme and hence breaks the traceability property. For a Type-c2 adversary, let cmt_1 and cmt_2 be the commitments in ot' and rt', respectively. Since these were

both recycled from previous token queries, let $\mathsf{res}_1, \mathsf{op}_1$ and $\mathsf{res}_2, \mathsf{op}_2$ be the resources and opening information for which they were generated by the oracles at the time. We can run the knowledge extractor for the NIZK PoK lt' to extract witnesses $\mathsf{res}', \mathsf{op}'_1, \mathsf{op}'_2$ that prove that $\mathsf{cmt}_1, \mathsf{cmt}_2$ are commitments to the same message. Since $\mathsf{res}_1 \neq \mathsf{res}_2$ due to the winning condition of a Type-c adversary, we know that $\mathsf{res}' \neq \mathsf{res}_i$ for at least one of $i \in \{1, 2\}$. Therefore, $(\mathsf{res}_i, \mathsf{op}_i)$ and $(\mathsf{res}', \mathsf{op}'_i)$ are two valid openings for the same commitment cmt_i, thereby breaking the binding property of the commitment scheme. □

Theorem 4. *If the commitment equality proof is complete and the group signature scheme is traceable, then our construction satisfies the provider security property.*

Sketch. There are six different reasons for the $\mathsf{Extract}$ algorithm to output $\perp$, namely:

1. $\mathsf{GVerify}(\mathsf{gpk}, (\texttt{policy}, c, \mathsf{cmt}_1, \mathsf{otpk}, \mathcal{SP}), \sigma_1) = 0$,
2. $\mathsf{OTVerify}(\mathsf{otpk}, (c, \mathsf{cmt}_1, \sigma_1), \mathsf{ots}) = 0$,
3. $\mathsf{GVerify}(\mathsf{gpk}, (\texttt{access}, \mathsf{cmt}_2, \mathcal{SP}), \sigma_2) = 0$,
4. verification of the NIZK PoK lt fails w.r.t. $\mathsf{cmt}_1, \mathsf{cmt}_2$, or
5. one of the recovered identities i, j is $\perp$.

Reasons 1 and 2 cannot occur since they would have caused the OTVf to reject the owner token, while reason 3 would have caused the LTGen to return $\perp$. Reason 4 contradicts the completeness of the commitment equality proof. Finally, reason 5 means that a policy or requester token contains a valid group signature that cannot be opened, which violates the traceability of the group signature scheme. □

4 Implementation with OAuth

In the following, we discuss how an existing SNS can be extended to be an SNS-based access control system by means of existing web technology. As we will see, we are able to achieve most of our envisioned security requiremens by utilizing the OAuth 2.0 authorization framework [17]. However, linking tokens to a particular resource is not part of this solution. In the following, we first introduce OAuth, then we give the implementation details, and afterwards we informally discuss the implementation's security properties.

4.1 OAuth Authorization Framework

OAuth 2.0 enables users to authorize third-party access to their online resources by providing a so-called *OAuth access token* to the third party instead of sharing their actual access credentials (such as their username and password). For example, a user could grant some photo printing service limited access to her Google+ image gallery by providing the printing service with an access token. Such a token is an opaque string that represents an access authorization issued to the bearing third party. The third party making requests to protected resources on behalf of the user is also called a *client* in OAuth. OAuth tokens are issued to clients (e.g., $\mathcal{SP}$) by an authorization server (e.g. $\mathcal{SN}$) with the approval of the resource owner (e.g. some user $\mathcal{U}_i$). For security reasons (cf. [17, Sec. 3.2.1.]), clients have to register with the authorization server to obtain authentication credentials consisting of a client identifier and password.

To discuss the security properties of our implementation later on, in the following we briefly outline the five steps of the most common OAuth message flow [17]:

1. The client initiates the protocol flow by (re-)directing the resource owner's user agent (e.g., her web browser) to the authorization server. The client includes its client identifier, the requested access *scope*, and a URI to which the authorization server will redirect the user back once access is granted (or denied).
2. The authorization server authenticates the resource owner via the user agent and establishes whether she grants or denies the client's access request for the given scope.
3. In case the resource owner grants access, the authorization server redirects the user agent back to the client using the redirection URI provided earlier. The redirection URI includes an *authorization code*.
4. The client requests an access token from the authorization server by providing the server with the authorization code received in the previous step. When making the request, the client includes her authentication credentials obtained during registration (e.g., with HTTP Basic authentication). The request is made using a server-authenticated channel with Transport Layer Security (TLS).
5. The authorization server authenticates the client and validates the authorization code. If valid, the authorization server responds back with an access token.

4.2 Implementation

In this section, we describe how an SNS $\mathcal{SN}$ can implement a web service endpoint (e.g., a RESTful API) to be used by $\mathcal{SP}$ that on input of an owner token ot and an requester token rt returns a boolean answer on whether the policy associated with ot as evaluated under the user identity associated with ot is satisfied w.r.t. the user identity associated with rt, i.e., $0/1 \leftarrow \mathsf{policyEval}(\mathsf{ot}, \mathsf{rt})$. We realize rt and ot as OAuth access tokens and use the OAuth protocol as previously described to provide $\mathcal{SP}$ with these tokens. We assume that $\mathcal{SP}$ has registered with $\mathcal{SN}$ as OAuth client and obtained authentication credentials accordingly. $\mathcal{SN}$ may choose to grant access to its endpoint only to parties who authenticate with their OAuth client credentials.

4.2.1 Resource Deposition

For a user $\mathcal{U}_i$ to be allowed to deposit some resource with description res (e.g., the resource URL) with $\mathcal{SP}$, she has to provide $\mathcal{SP}$ with an owner token ot. With this token, $\mathcal{U}_i$ authorizes $\mathcal{SP}$ to evaluate some policy plc on her behalf by means of the mentioned service endpoint. To obtain this token, $\mathcal{SP}$ executes the OAuth authorization protocol as client with $\mathcal{U}_i$ and with $\mathcal{SN}$ as authorization server. $\mathcal{SP}$ uses the scope *policyEval_owner* to unambiguously indicate to $\mathcal{SN}$ and to the user that it requests the generation of an owner token that it wants to use for policy evaluation with $\mathcal{SN}$'s service endpoint. During user authentication in step 2, $\mathcal{U}_i$ provides $\mathcal{SN}$ not only with her credentials (e.g., username and password), but also with the policy plc that shall be associated with the owner token (e.g., by filling out some web form). Before $\mathcal{SN}$ returns ot in step 5 of the OAuth flow, it first generates this token as random string with sufficient entropy and associates it in some local database with the authenticated user $\mathcal{U}_i$, the policy plc, and the requested scope. Having received the owner token ot, $\mathcal{SP}$ allows to deposit the resource and associates it with res and ot in some local database. Having deposited the resource, $\mathcal{U}_i$ sends res to her friends (e.g., by email) so that they can access the resource.

4.2.2 Resource Access

When some user $\mathcal{U}_j$ wants to access the resource with description res hosted by $\mathcal{SP}$, the user first has to provide $\mathcal{SP}$ with a one-time requester token rt so that $\mathcal{SP}$ can make a policy evaluation query. With this token, $\mathcal{U}_j$ authorizes $\mathcal{SP}$ to involve her in a policy check done with $\mathcal{SN}$'s service endpoint. To obtain the token, $\mathcal{SP}$ executes the OAuth protocol as client with $\mathcal{U}_j$ and with $\mathcal{SN}$. $\mathcal{SP}$ uses the scope *policyEval_requester* to indicate to $\mathcal{SN}$ and to the user that it requests the generation of a requester token that it wants to use for policy evaluation with $\mathcal{SN}$'s service endpoint. Before $\mathcal{SN}$ returns rt in step 5 of the OAuth flow, it first generates this one-time token as random string with sufficient entropy and associates it in some local database with both the authenticated user $\mathcal{U}_j$ and the requested scope. After $\mathcal{SP}$ has obtained the requester token, it retrieves the owner token associated with res and queries $\mathcal{SN}$'s policy evaluation endpoint (e.g., by means of a HTTP GET request) by using the retrieved owner token and the obtained requester token as query parameters.

4.2.3 Policy Evaluation

Upon receiving a policy evaluation query on its web service endpoint from $\mathcal{SP}$, $\mathcal{SN}$ retrieves from its local database the user identities $\mathcal{U}_i$ and $\mathcal{U}_j$ and the scopes s_o and s_r that are associated with the provided owner and requester token, respectively. It also retrieves the policy plc associated with the owner token from this database. Afterwards, $\mathcal{SN}$ checks whether s_o and s_r equal *policyEval_owner* and *policyEval_requester*, respectively. If the tokens are valid, the scopes match, and the requester token has not been used before, $\mathcal{SN}$ evaluates plc under $\mathcal{U}_i$'s identity (i.e., as if $\mathcal{U}_i$ would execute the query herself while being logged in) which query results in a set of user identities. Finally, $\mathcal{SN}$ responds to $\mathcal{SP}$ in a boolean fashion whether $\mathcal{U}_j$ is member of this set and marks rt as used. If one of the tokens is not valid or the scopes do not match, $\mathcal{SN}$ aborts the transaction. If the service request was authenticated, $\mathcal{SN}$ may optionally deny the authenticated party future access to the service endpoint.

4.3 Security Properties

Here we discuss the security properties of our implementation based on the OAuth protocol. Because the security requirements are defined with respect to the procedures defined in Sec. 2.2, we discuss the properties only informally and do not elaborate on the correctness of the implementation.

Anonymity Assuming that both the owner token and the requester token are opaque strings of sufficient length that are chosen randomly by $\mathcal{SN}$ with sufficient entropy, these strings neither reveal the user identities nor the policy that are associated with the tokens. The only place where the relation between token strings and the associated user identities and policy are stored, is the local database of $\mathcal{SN}$. We assume that this database is accessible only by $\mathcal{SN}$. Further, the OAuth message flow for providing $\mathcal{SP}$ with an access token ensures that both the user's identity in $\mathcal{SN}$—and the policy when setting up an owner token—are confidentially communicated to (and only to) $\mathcal{SN}$ in step 2 of the flow. Thus, $\mathcal{SP}$ cannot tell which identity is associated with a token, or even whether two tokens were generated by the same or by different users. It follows that our implementation of an SNS-based access control system with OAuth tokens as described above satisfies the anonymity requirement.

Note that we assume that both the owner token and the requester token provided to $\mathcal{SP}$ have *only* the OAuth scopes *policyEval_owner* and *policyEval_requester*, respectively. In case the tokens

have also other access scopes, $\mathcal{SP}$ may be able to extract the identities of the associated users. For example, in Facebook, every OAuth token allows a client to query both the public profile and the friend list of the associated user, independent of which (additional) scopes the token has. The public profile information includes the user's identifier, first name, last name, username, and gender. Thus, if Facebook were to implement our scenario as described above, the basic permissions of Facebook OAuth tokens would have to be adapted such that the public profile and the friend list can only be accessed with appropriate access scopes.

Resource Secrecy Formally, resource secrecy requires that no token leaks information to $\mathcal{SN}$ about the resource for which the tokens were generated. Since owner and requester tokens are generated by $\mathcal{SN}$ itself, they cannot leak any information to $\mathcal{SN}$ about the resource for which the tokens were generated. Adapted to our OAuth scenario, resource secrecy requires that $\mathcal{SN}$ does not learn which resource is associated with an owner token, and which resource is being accessed by means of a requester token. Given that neither of the OAuth message flows for providing $\mathcal{SP}$ with an owner or requester token involve any information about the resource that is deposited or accessed, the resource secrecy requirement is satisfied.

Token Unforgeability Token unforgeability asks that a cheating service provider cannot modify the policy set by the resource owner, and that a cheating user cannot get access if she does not satisfy the policy. Assuming that a cheating service provider $\mathcal{SP}$ does not have access to the database of $\mathcal{SN}$ that stores the relation between owner token strings and policies, $\mathcal{SP}$ cannot modify the policy set by the resource owner. Further, assuming that $\mathcal{SN}$-users cannot modify profile information of other users (e.g., their friends, or friend lists), the policy evaluation procedure of $\mathcal{SN}$, or $\mathcal{SP}$'s procedure for granting access based on the results of a policy evaluation query, and assuming that the policy evaluation result is communicated securely (e.g., by using a TLS connection), users who do not satisfy the policy cannot get access. If a policy only concerns profile information of one user, then a cheating user who knows the policy could modify his own profile information such that it matches the policy.

Token unforgeability further requires that $\mathcal{SN}$ can check whether an owner token and a requester token are created for the same resource, but at the same time, resource secrecy mandates that $\mathcal{SN}$ should not learn any information about the resource. These requirements are rather difficult to reconcile with our OAuth implementation. One could associate tokens in $\mathcal{SN}$'s system with the hash values of the resource identifier or its content, but this requires the resources or their identifiers to have high entropy, and certainly does not meet our strong notion of indistinguishability. We therefore cannot prevent that $\mathcal{SP}$ reuses a requester token to test whether the requester also satisfies policies associated to other resources than the one he queried. This can be somewhat mitigated by letting $\mathcal{SN}$ accept each requester token only once for policy evaluation.

Provider Security Provider security requires that no adversary is able to provide $\mathcal{SP}$ with invalid access tokens. Because access tokens are produced by $\mathcal{SN}$ itself in its role as authorization server and obtained by $\mathcal{SP}$ via a TLS connection with server authentication by means of a one-time authorization code without passing through the resource owner's user agent, $\mathcal{SP}$ can be sure that the tokens are indeed coming from $\mathcal{SN}$ and not from an adversary. Thus, provider security is satisfied.

5 Efficiency Discussion

The efficiency of our constructions based on group signatures mainly depends on the chosen instantiations for the underlying building blocks. There are many trade-offs to be made, e.g., of efficiency versus security assumptions, and of bandwidth versus computation. One slight problem is that some of the most efficient group signature schemes [6, 4] have opening linear in the number of group members, i.e., the number of users in the social network, which would be prohibitive in our case. However, the owner and requester can encrypt their usernames to $\mathcal{SN}$ and include the ciphertext in the owner and requester tokens, respectively. This comes at negligible extra cost in the owner token because the username can be encrypted together with the policy, and comes at the cost of an additional ciphertext in the requester and linking token. As an illustrative example, let's consider our construction when instantiated using elliptic curves with the group signature scheme by Bichsel et al. [4], ElGamal encryption [12] in combination with the Fujisaki-Okamoto transformation [15] to obtain IND-CCA2 security, and Schnorr signatures [22]. The generation of an owner token in such an instantiation takes 9 exponentiations, the verification takes 2 pairings and 4 exponentiations. The owner token itself contains 8 group elements and 3 exponents, taking 2816 bits for a 256-bit curve. Generation of a requester token takes 7 exponentiations, while the token itself contains 6 group elements and 2 exponents, or 2048 bits. Linking token generation takes 2 pairings and 5 exponentiations, the token itself contains 4 exponents or 1024 bits. Finally, extraction takes 10 pairings and 13 exponentiations.

Our OAuth implementation is very efficient because the required operations merely involve simple HTTP redirects, the inexpensive creation and lookup of a few database entries, the generation of two secure random numbers that represent the tokens, and $\mathcal{SN}$'s evaluation of the policy under $\mathcal{U}_i$'s identity. The latter represents a query that results in a set of user identities, and such queries are highly optimized by today's SNS providers because they are also used to search for new friends in an SNS. Facebook's newly introduced *Graph Search* is a prominent example of such a query possibility.

In terms of number of communication flows, the group signature construction is more efficient than the OAuth construction. Namely, our group signature construction only requires a single communication flow to deposit a resource at the SP, while accessing a resource involves two rounds of communication. The OAuth scheme, on the other hand, requires five HTTP request-response transactions to deposit a resource, and needs six such transactions to access a resource.

6 Conclusion

We presented the first privacy-preserving mechanisms to leverage profile and relationship information maintained in SNSs to make access control decisions for externally hosted resources. We presented two constructions: the first is a novel cryptographic scheme based on group signatures, the second uses the standardized OAuth authorization protocol. The OAuth protocol is computationally more efficient because it only requires random number generation and database lookups, as opposed to several public-key operations for the group signature scheme. On the other hand, the group signature construction has a simpler communication flow with less redirections and message round-trips between the participants. In terms of security, the group signature construction has the advantage that owner and requester tokens are tightly bound to a particular resource, preventing a cheating service provider from reusing the requester token to test whether the requester also satisfies the policy attached to a different resource. Such protection cannot be achieved with OAuth,

but the effect can be mitigated by letting the SNS limit requester tokens to be used only once in a policy evaluation query.

The group signature construction is also more amenable to further cryptographic extensions. For example, it could be extended such that the owner of the resource is informed which users accessed the resource, without $\mathcal{SP}$ and $\mathcal{SN}$ learning more information than they do in the current scheme. In a nutshell, this could be realized by also generating public keys for users as part of the registration protocol and letting $\mathcal{SN}$ store the public keys. Then, whenever $\mathcal{SN}$ gets a request from $\mathcal{SP}$ to extract (i, j, plc) from $(\mathsf{ot}, \mathsf{rt}, \mathsf{lt})$ and to evaluate the policy plc, $\mathcal{SN}$'s reply could not only say whether access should be granted, but also contain an encryption of j under $\mathcal{U}_i$'s public key. $\mathcal{SP}$ can then store all these encryptions and provide them to $\mathcal{U}_i$ upon request. We leave the details of a concrete realization as well as a formal definition of the required additional procedures and security properties as future work.

Another possible useful extension of our scheme would be to allow the definition of policies spanning several social networks in such a way that the different accounts at the different social networks are guaranteed to be held by the same user, and such that it is not leaked which parts of a policy a user satisfies (e.g., the part with respect to one social network $\mathcal{SN}_1$, but not with another $\mathcal{SN}_2$).

The access control mechanisms presented in this paper essentially let the owner of the resource delegate to $\mathcal{SP}$ the right to perform policy evaluation queries at $\mathcal{SN}$ with respect to the owner's identity. Additionally, the requester's token acts as her permission to evaluate the policy on her identity. One possible extension would be to make the requester aware of the policy that will be evaluated. To keep the policy hidden from $\mathcal{SP}$, it would likely have to be encrypted under the user's public key. As another extension, the social network could impose additional restrictions on which type of policies the resource owners can associate to their resources, or let requesters indicate in their privacy settings which policies can be evaluated on their profiles. For example, it would make sense that $\mathcal{SN}$ imposes that the owner's policy cannot take into account any information about other users that the owner doesn't have access to on the social network. Alternatively, one could let the requester specify his privacy preferences as part of the requester token.

7 Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme for the projects *ABC4Trust* (grant agreement no. 257782), *FutureID* (grant agreement no. 318424), and *PERCY* (grant agreement no. 321310).

References

[1] M. Bellare, D. Micciancio, B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. *EUROCRYPT 2003*, Springer, 2003.

[2] M. Bellare, P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *ACM CCS 1993*, ACM, 1993.

[3] M. Bellare, H. Shi, C. Zhang. Foundations of group signatures: The case of dynamic groups. *CT-RSA 2005*, Springer, 2005.

[4] P. Bichsel, J. Camenisch, G. Neven, N. Smart, B. Warinschi. Get Shorty via Group Signatures without Encryption. SCN 2010, Springer, 2010.

[5] M. Blum, P. Feldman, S. Micali. Non-Interactive Zero-Knowledge and Its Applications. *STOC 1988*, ACM, 1988.

[6] D. Boneh, H. Shacham. Group Signatures with Verifier-Local Revocation. *ACM CCS 2004*, ACM, 2004.

[7] J. Camenisch, A. Kiayias, M. Yung. On the Portability of Generalized Schnorr Proofs. *EUROCRYPT 2009*, Springer, 2009.

[8] R. Canetti, S. Halevi, J. Katz. Chosen-ciphertext security from identity-based encryption. *EUROCRYPT 2004*, Springer, 2004.

[9] B. Carminati, E. Ferrari. Privacy-Aware Access Control in Social Networks: Issues and Solutions. *Privacy and Anonymity in Information Management Systems*, Springer, 2010.

[10] B. Carminati, E. Ferrari, A. Perego. Enforcing access control in Web-based social networks. ACM TISSEC, 13 (1):6, 2009.

[11] D. Chaum, E. van Heyst. Group signatures. *EUROCRYPT 1991*, Springer, 1991.

[12] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory 31(4):469–472, 1985.

[13] A. Fiat, A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *CRYPTO 1986*, Springer, 1987.

[14] P.W.L. Fong. Relationship-Based Access Control: Protection Model and Policy Language, In *ACM CODASPY*, ACM, 2011.

[15] E. Fujisaki, T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *CRYPTO 1999*, Springer, 1999.

[16] C.E. Gates. Access control requirements for Web 2.0 security and privacy. *IEEE Web 2.0 Privacy and Security Workshop (W2SP'07)*, IEEE, 2007.

[17] D. Hardt. The OAuth 2.0 Authorization Framework. IETF RFC 6749, 2012.

[18] H. Hu, G.-J. Ahn, J. Jorgensen. Multiparty Access Control for Online Social Networks: Model and Mechanisms. IEEE TKDE, 25 (7):1614–1627, 2013.

[19] L. Lamport. Constructing Digital Signatures from a One-Way Function. TR CSL-98, SRI, 1979.

[20] P. MacKenzie, M. Reiter, K. Yang. Alternatives to non-malleability: Definitions, constructions, and applications. *TCC 2004*, Springer, 2004.

[21] T. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. *CRYPTO 1991*, Springer, 1991.

[22] C.-P. Schnorr. Efficient Identification and Signatures for Smart Cards. *EUROCRYPT 1989*, Springer, 1989.

[23] V. Shoup. A proposal for an ISO standard for public key encryption (version 2.1). Manuscript, 2001.

[24] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Manuscript, 2004 (revised 2006).