A Credential-Based System for the Anonymous Delegation of Rights

Liesje Demuynck[⋆], Bart De Decker, and Wouter Joosen

Katholieke Universiteit Leuven, Department of Computer Science,
Celestijnenlaan 200A, 3001 Heverlee, Belgium
{Liesje.Demuynck,Bart.DeDecker,Wouter.Joosen}@cs.kuleuven.be

Abstract. An anonymous delegation system enables individuals to retrieve rights and to delegate different subparts of these rights to different entities. The delegation procedure is anonymous, such that no collusion of entities can track an individual's delegation behavior. On the other hand, it is ensured that a user cannot abuse her delegation capabilities. This paper introduces a general delegation model and presents an implementation. Our implementation is based on credential systems and provides both anonymity for the individual and security for the organizations.

1 Introduction

The concept of authentication and authorization has long been studied in computer science. Intuitively, all solutions follow the same procedure: the user first retrieves her access rights from a trusted authority and afterwards shows it to a service provider. For security reasons, the retrieval protocol will typically be performed in an identified or pseudonymous manner. The showing protocol, on the other hand, may be performed in an anonymous but controlled fashion: users are anonymous but can still be held accountable for their actions [2, 4].

In many applications, the owner of a right may need to delegate (part of) her right to a different entity. Consider, for example, a doctor having access to a medical database. When she is absent from the hospital, she may grant one of her assistants access to some specific files in the database. She may prefer this delegation procedure to be anonymous, such that no central authority can monitor her delegation behavior. On the other hand, it should be ensured that she cannot abuse her delegation capabilities in any way.

Wohlgemuth et al. [12] present a privacy-preserving delegation system in the context of business processes with proxies; a user delegates some of her rights to a proxy, who may then use these rights to access services on the user's behalve. The authors do not assume a delegate to be anonymous. In addition, a lot of trust is put in a central certification authority, who knows what subrights are issued and to which proxies. Finally, re-delegation is not achieved.

[⋆] Ph. D. fellowship of the Research Foundation - Flanders (FWO).

This paper introduces a formal model for a delegation system and presents an implementation based on anonymous credentials. Our model can be used in various applications and achieves, among others, controlled re-delegation and the revocation of rights. Anonymity is provided for the delegator as well as for the delegate. At the same time, the security of individuals and service providers is protected and users can be held accountable for their actions.

The outline of this paper is as follows. Section 2 presents a formal model for the delegation system. Section 3 introduces the basic building blocks for the implementation: commitments, anonymous credentials and verifiable encryptions. The system itself is described in Section 4 and evaluated in Section 5. We conclude in Section 6.

2 General delegation model

We first present a general model for the anonymous delegation of rights. Section 2.1 gives a global overview of the system's entities and protocols. Section 2.2 then states some assumptions on the behavior of these entities and Section 2.3 describes a general set of requirements on the system's behavior.

2.1 Roles and protocols

Roles. An entity in the system is either a *user U* or an *organization O*.

An organization must at all times be identifiable. It is either a *registrar RG*, an *issuer I*, a *verifier V* or a *revocation manager RM*. A registrar registers users to the system and an issuer issues rights to these users. A right contains a set of specifications and a validity period. It can be shown to a verifier or it can be used to issue sub-rights. These sub-rights can in turn be used to issue sub-rights of themselves. As such, a *delegation tree* of a right is constructed. The root of this tree is the right itself, while all other nodes are sub-rights of their parent-node. When abuse of a right is detected, or when it is no longer needed, the right as well as all other rights in its delegation tree, are revoked by the revocation manager.

In contrast to an organization, a user may be anonymous within the system. It can be either a *delegator Do* or a *delegate De*. Do delegates part of her right to De. We will refer to Do's right as the main-right and to De's new right as the corresponding sub-right. Note that a right can be both a main-right w.r.t. one right and a sub-right w.r.t. another right. (e.g. an access right to sections $\{A, B\}$ of a database may be a sub-right w.r.t. an access right to sections $\{A, B, C\}$, and a main-right w.r.t an access right to section $\{A\}$). Similarly, a user can be both a delegator and a delegate with respect to different users in the system.

Protocols. A summary of the system protocols is given in Table 1.

U registers to the system by performing the *Registration* protocol with RG. She retrieves a right `R` satisfying specifications `RSpecs` by performing the *IssueRight* protocol with issuer I. As a result, I receives a transcript `IssueTrans`.

Table 1. general delegation model - protocol overview.

$U \leftrightarrow RG$	: *Registration*(certifications)
$I \leftrightarrow U$	: *IssueRight*(RSpecs) returns R ; IssueTrans
$Do \leftrightarrow De$	: *DelegateRight*([I], MR, SRSpecs) returns SR; IssueTrans
$U \leftrightarrow V$	: *ShowRight*(R, showProperties)
RM	: *RevokeRight*(revTag)

The *DelegateRight* protocol takes as input both a main-right `MR` and a specification `SRSpecs` of the new sub-right. It outputs a transcript `IssueTrans` for delegator *Do* and a sub-right `SR` for delegate *De*. Potentially, an additional issuer *I* may be involved in the protocol.

A right `R` can be shown to *V* by means of the *ShowRight* protocol. Attribute `showProperties` specifies the right's properties which are revealed to *V*. Note that this may be only a subpart of the entire right. As an example, consider a right granting full database access to *U*. When showing this right to *V*, *U* may decide to only reveal her access rights for a particular subpart of the database.

Finally, a right can be revoked by means of the *RevokeRight* protocol. The input to this protocol is a revocation tag `revTag`. This tag can be found as a unique subpart of the `IssueTrans` transcript.

2.2 Assumptions

We employ the following assumptions concerning the entities in the system.

- System registrar and organizations can be trusted to perform their tasks correctly, i.e. they follow the protocols. This is a reasonable assumption and can, for example, be enforced by collecting secure logs of the parties' activities.
- All entities in the system can freely exchange their information. In particular, users may exchange information about the rights they have received. Note, however, that entities will not give away any information of which the secrecy is important to themselves. Examples of such information are secret keys and revocation information of sub-rights issued by themselves.

2.3 Requirements

We consider anonymity and security requirements. Anonymity requirements are optional and provide the user with a set of privacy guarantees. Security requirements are mandatory and protect the organization from malicious users.

Anonymity and linkability requirements.

A1. *Privacy preserving show protocol.* The *ShowRight* protocol should not reveal more information than what is absolutely necessary to gain access to *V*'s services. In particular, the following requirements should be satisfied.
   (a) *Anonymity.* Service access is anonymous.
   (b) *Unlinkability.* Different service accesses based on the same right cannot be linked to each other.

(c) *Right indistinguishability.* The access protocol does not reveal any information on how the access right was obtained.

A2. *Sub-right unlinkability.* Different sub-rights deduced from the same main-right must not be linkable to each other, even when all parties in the system (except for the main-right owner) share their information. This ensures that a user's delegation behavior cannot be tracked by other entities.

Security requirements.

S1. *Unforgeability.* Users may not successfully show a right which was not retrieved by means of an *IssueRight* or of a *DelegateRight* protocol.

S2. *Correct sub-rights.* The set of rights which are encoded in a sub-right must be a subset of the set of rights encoded in its corresponding main-right. In addition, the validity periods of a sub-right must fall within the validity period of its corresponding main-right.

S3. *Non-transferability.* The legitimate owner of a right must not be able to pass on the digital tokens constituting her right. Note that this requirement does not forbid to pass on a right by the delegation of a sub-right identical to the original right.

S4. *Consistency of rights.* Users may not be able to pool their rights in order to gain an asset (e.g. the access to a service or a new right), which each of them separately could not have obtained by correctly executing the protocols.

S5. *Correct revocation.* Rights must be revocable and the revocation of a right must include the revocation of all the rights in its delegation subtree. In addition, users must be prohibited to request the revocation of rights which are not issued or owned by themselves.

S6. *Conditional deanonymization.* In case of abuse of a right, appropriate measures should be taken against its owner. We distinguish two types of actions.
- retrieval of the owner's identity.
- retrieval of the right's issue transcript, enabling the right's revocation.

3 Basic building blocks

Our construction is based on commitments, credential systems and verifiable encryptions. We briefly introduce these concepts and their primitives. All communication is performed over anonymous communication channels.

Commitments. A commitment [11, 7] can be seen as the digital analogue of a "non-transparent sealed envelope". It enables a committer to hide a set of attributes (non-transparency property), while at the same time preventing her from changing these values after commitment (sealed property). The primitive

$$E : \mathit{Comm}, \mathit{OpenInfo} = \mathit{Comm}(\{\texttt{attrName} := \mathit{attrValue}, \ldots\})$$

enables an entity E to create a commitment *Comm* on a set of attributes. Additionally, she retrieves a secret key *OpenInfo* containing, among others, the attributes encoded into *Comm*. This key can be used to prove properties concerning the attributes.

$$E_1 \rightarrow E_2 : \mathit{ComProps}(\mathit{Comm}, P(\texttt{attr1}, \ldots))$$

The public input to this protocol is both a commitment *Comm* and a boolean predicate P concerning *Comm*'s attributes. For example, P may be the predicate ($\texttt{attr}_1 > 0$). If E_2 accepts, she is convinced that E_1 knows the *OpenInfo* belonging to *Comm*, and that *Comm*'s attributes satisfy predicate P. She does not find out any other information concerning *Comm* or *Comm*'s attributes.

Credentials. A credential system [2, 4] allows for anonymous yet accountable transactions between users and organizations. In the remainder of the paper, we employ the system proposed by Camenisch et al. [4, 1, 6].

A credential *Cred* is retrieved from I by means of the *CredGet* protocol.

$$U \leftarrow I : \mathit{Cred} = \mathit{CredGet}(\{\texttt{attr}_1 := G(.), \ldots\})$$

It consists of a set of attributes as well as a secret key for showing it to a verifier.

Each attribute is constructed as a separate function $G(.)$ of public values and attributes encoded into previously shown credentials or commitments. As an example, $\texttt{attr}_1$ may be constructed as $\texttt{attr}_1 := \mathit{Cred}_x.\texttt{a}_1 + 5$, where $\mathit{Cred}_x.\texttt{a}_1$ refers to attribute $\texttt{a}_1$ of a previously shown credential Cred_x. Issuer I cannot find out any information concerning the credential's attributes, apart from the fact that they are constructed correctly based on $G(.)$.

During the *CredShow* protocol, U shows her credential *Cred* to V.

$$U \rightarrow V : \mathit{CredShow}(\mathit{Cred}, P(\texttt{attr}_1, \ldots))$$

Additionally, U reveals a boolean predicate P concerning public values, attributes occurring in *Cred* and attributes occurring in previously shown credentials or commitments. For example, P may be the predicate ($\texttt{attr}_1 > C_x.\texttt{a}_1 \;\wedge\; \texttt{attr}_1 < C_x.\texttt{a}_2$), where $C_x.\texttt{a}_1$ and $C_x.\texttt{a}_2$ refer to attributes $\texttt{a}_1$ and $\texttt{a}_2$ encoded into a previously shown commitment C_x. V cannot learn any new information from the execution of the protocol, apart from the fact that U has a valid credential which is issued by I and of which the attributes satisfy P.

Different show-protocols of the same credential cannot be linked to each other, nor can they be linked to their issue protocol.

Using the *CredSign* protocol, a credential can be used to sign a message.

$$U : \mathit{Sig} = \mathit{CredSign}(\mathit{Cred}, P(\texttt{attr}_1, \ldots), \mathit{msg})$$

The properties of this protocol are exactly the same as for the *CredShow* protocol, except for the additional fact that a message *msg* is signed using the credential. For ease of representation, we assume that output *Sig* contains the signature as well as the signed data.

Verifiable encryptions. Verifiable encryptions [5] have all the characteristics of regular encryptions. Based on a public key pk, any user U can encrypt a message. In addition, U can demonstrate properties of the encrypted plaintext. For example, U can prove to V that the encrypted plaintext is encoded as an attribute in a previously shown credential or commitment. This is denoted as a predicate $c = VE(\mathtt{x})$, where c refers to the ciphertext and where $\mathtt{x}$ refers to the credential's (or commitment's) attribute.

Note that c is created using a public key pk of which the corresponding secret key sk may not be known by V. For ease of representation we omit the specifications of pk and its owner. We merely assume its owner to be an entity T which can be contacted when decryption is needed. Additionally, T is trusted not to perform any unwanted decryptions.

The use of credentials, commitments and verifiable encryptions offers numerous advantages in the construction of privacy-sensitive applications. Credentials are unforgeable and allow for service accesses which are anonymous and unlinkable. By combining them with commitments and verifiable encryptions, additional properties such as non-transferability, consistency of credentials, conditional deanonymization and revocation can easily be added using standard techniques [2, 4]. These properties will turn out to be very handy in our final construction.

4 The delegation system

We first give a general outline of the system and its components. Afterwards, the system and its protocols are described in more detail.

4.1 General outline of the system

All rights in the system are represented as digital credentials. In particular, main-rights and sub-rights have an identical credential structure. The credential's attributes consist of a tuple $(id, e, RSet)$, where id is the owner's identity, e is the right's revocation tag and $RSet$ is a specification of the right. A sub-right is issued by constructing a new tuple $(id', e', RSet')$ and by signing a commitment on this tuple. Note that we sign a commitment rather than the actual tuple $(id', e', RSet')$. This way the tuple is hidden from any third parties. The signature is created by the main-right's credential and by using the *CredSign* protocol. In a final step, this signature is exchanged with I for a new credential.

To achieve correct revocation, the sub-right's revocation tag e' may not be arbitrarily chosen by Do. Instead, it must be requested from RM through an auxiliary *IssueRevTags* protocol. During the protocol, Do retrieves a credential $Cred_{rev}$ containing e and a list of random revocation tags. RM does not know the values of these tags, but she is able to recover them as soon as the corresponding main-right is revoked. Furthermore, by means of the attribute value

e occurring in both credentials, $Cred_{rev}$ is invisibly bound to the main right's credential.

An example is given in Figure 1. Doctor Jones has access to sections A, B and C of the hospital's database. This is represented by a credential $Cred_{ABC}$ containing a revocation tag e. In addition, she owns a credential $Cred_{rev}$ which is retrieved during an *IssueRevTags* protocol. $Cred_{rev}$ encodes $Cred_{ABC}$'s revocation tag e. As such, it can only be used to delegate sub-rights based on $Cred_{ABC}$. In addition, $Cred_{rev}$ contains a set $(e_1, \ldots, e_n)$ of random revocation tags. Whenever a sub-right is issued, its new revocation tag must be one of these values e_i encoded into $Cred_{rev}$. In our example, Dr. Jones has delegated two sub-rights based on $Cred_{ABC}$. Due to the randomness of the e_i's, their corresponding credentials cannot be linked to each other.

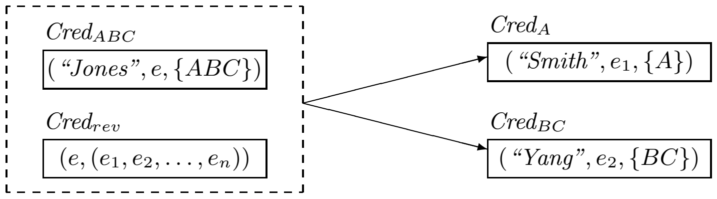
$Cred_{ABC}$
$(\text{"Jones"}, e, \{ABC\})$
$Cred_{rev}$
$(e, (e_1, e_2, \ldots, e_n))$
$Cred_A$
$(\text{"Smith"}, e_1, \{A\})$
$Cred_{BC}$
$(\text{"Yang"}, e_2, \{BC\})$

Fig. 1. Example credential structures

4.2 Protocol description

The delegation system is depicted in Figures 2 and 3. We now give a detailed description of the protocols.

Registration. U registers to the system by authenticating to *RG*. She then receives a credential $Cred_u$ containing her global identifier id_u. In the remainder of the paper, we will refer to this credential as $Cred_{do}$, $Cred_{de}$ or $Cred_u$, depending on its owner's role as a delegator, a delegate or a user.

IssueRight. U first proves to be registered to the system. If successful, she retrieves a credential $Cred_{right}$ containing three attributes: a copy of attribute `id` in $Cred_u$, an issuer-chosen revocation tag e and a specification *RSet* of rights. We will refer to this credential as $Cred_{right}$, $Cred_{main}$ or $Cred_{sub}$, depending on its function as a general right (which can be both a main-right or a sub-right), a main-right or a sub-right.

IssueRevTags. This protocol can be executed multiple times for the same value e. It provides *Do* with n additional revocation tags for her sub-rights. First, *Do* creates a commitment C_{rev} containing random values $e_{1u}, \ldots, e_{nu}$. This commitment, together with the main-right's revocation tag e and a verifiable encryption *encr* of $(e_{1u}, \ldots, e_{nu})$ are sent to *RM*. *Do* also proves that *encr* is constructed correctly. After receiving random values $e_{1i}, \ldots, e_{ni}$ from *RM*, *Do* proves that value e is the same revocation tag as is encoded in her main-right. For this, she creates a credential-based signature Sig_{rev}. This ensures that *RM* is provided with sufficient evidence of the transaction. Finally, when

• **Registration.**

$$U \xrightarrow{\quad \textit{user authentication} \quad} RG$$

$$U \xleftarrow{\quad \mathit{Cred}_u = \mathit{CredGet}(\{\texttt{id} := id_u\}) \quad} RG$$

• **IssueRight.**

$$U \xrightarrow{\quad \mathit{CredShow}(\mathit{Cred}_u,\ \mathit{null}) \quad} I$$

$$U \xleftarrow[\{\texttt{Owner} := \mathit{Cred}_u.\texttt{id},\ \texttt{revTag} := e,\ \texttt{rights} := \mathit{RSet}\})]{\mathit{Cred}_{right} = \mathit{CredGet}(} I$$

• **IssueRevTags.**

Do: choose random values $e_{1u}, \ldots, e_{nu}$
$C_{rev}, O_{rev} = \mathit{Comm}(\{\mathtt{e_{1u}} := e_{1u}, \ldots, \mathtt{e_{nu}} := e_{nu}\})$
$\mathit{encr} = \mathit{VE}(e_{1u}, \ldots, e_{nu})$

$$Do \xrightarrow{\quad e, \mathit{encr}, C_{rev}, \mathit{ComProps}(C_{rev}, \{\mathit{encr} == \mathit{VE}(\mathtt{e_{1u}}, \ldots, \mathtt{e_{nu}})\}) \quad} RM$$

RM: choose random values $e_{1i}, \ldots, e_{ni}$

$$Do \xleftarrow{\quad e_{1i}, \ldots, e_{ni} \quad} RM$$

Do: $\mathit{Sig}_{rev} = \mathit{CredSign}(\mathit{Cred}_{right}, \{\texttt{revTag} == e\},$
$(e, e_{1i}, \ldots, e_{ni}, \mathit{encr}))$

$$Do \xrightarrow{\quad \mathit{Sig}_{rev} \quad} RM$$

$$Do \xleftarrow[\{\texttt{revTag} := e, \mathtt{e_1} := C_{rev}.\mathtt{e_{1u}} + e_{1i}, \ldots, \mathtt{e_n} := C_{rev}.\mathtt{e_{nu}} + e_{ni}\})]{\mathit{Cred}_{rev} = \mathit{CredGet}(} RM$$

Fig. 2. Credential-based implementation of the delegation model (1/2)

e has not been revoked, a credential Cred_{rev} is issued by RM. The attributes of this credential consist of e and of the new revocation tags $e_1, \ldots, e_n$ which are constructed as $e_k = e_{ki} + e_{ku}$ for $k = 1, \ldots, n$. Note that the resulting e_k's are unknown to RM; she only knows that they are constructed correctly as $e_k = C_{rev}.\mathtt{e_{ku}} + e_{ki}$. Moreover, their values cannot be manipulated by Do.

DelegateRight. This protocol consists of two phases which can be separated in time. It may be preceded by an optional identification step from De to Do.

During the first phase, De creates a commitment C_{de} on her global identifier id_{de}. She sends it to Do and proves that it is constructed correctly. Upon success, Do creates two commitments C_{do} and C_{sub}. C_{do} encodes her main-right's revocation tag e, while C_{sub} contains both the revocation tag e_i and the right-specifications SRSet of the prospective sub-right. Commitments C_{do}, C_{de} and C_{sub} are then signed by means of the *CredSign* protocol for credentials Cred_{main} and Cred_{rev}. This results in a signature tuple $(\mathit{Sig}_{sub1}, \mathit{Sig}_{sub2})$ which is sent with the key O_{sub} to De. The signatures ensure the following properties:

– The signer owns credentials Cred_{main} and Cred_{rev}.
– The same revocation tag e is encoded in both Cred_{main} and Cred_{rev}.
– The rights encoded into C_{sub} are a subset of the rights encoded into Cred_{main}.
– C_{sub}'s attribute `subRevTag` is one of the revocation tags encoded in Cred_{rev}.

• **DelegateRight.**

De: $C_{de},\ O_{de} = Comm(\{\texttt{id} := id_{de}\})$

$$Do \xleftarrow{\quad C_{de},\ CredShow(Cred_{de}, \{\texttt{id} == C_{de}.\texttt{id}\}) \quad} De$$

Do: $C_{sub},\ O_{sub} = Comm(\{\texttt{subRevTag} := e_i,\ \texttt{rights} := SRset\})$
$C_{do},\ O_{do} = Comm(\{\texttt{revTag} := e\})$
$Sig_{sub1} = CredSign(Cred_{main}, \{\texttt{revTag} == C_{do}.\texttt{revTag} \wedge \texttt{rights} \supset C_{sub}.\texttt{rights}\}, (C_{do}, C_{de}, C_{sub}))$
$Sig_{sub2} = CredSign(Cred_{rev}, \{\texttt{revTag} == C_{do}.\texttt{revTag} \wedge C_{sub}.\texttt{subRevTag} \in \{\texttt{e}_1, \ldots, \texttt{e}_n\}\}, (C_{do}, C_{de}, C_{sub}))$

$$Do \xrightarrow{\quad O_{sub},\ Sig_{sub1},\ Sig_{sub2} \quad} De$$

De: retrieve e_i and $SRset$ from O_{sub}
$Sig_{de} = CredSign(Cred_{de}, \{\texttt{id} == C_{de}.\texttt{id}\},\ (Sig_{sub1},\ Sig_{sub2}))$

$$I \xleftarrow{\quad Sig_{de} \quad} De$$

$$I \xrightarrow{\quad Cred_{sub} = CredGet(\{\texttt{owner} := C_{de}.\texttt{id},\ \texttt{revTag} := C_{sub}.\texttt{subRevTag}, \texttt{rights} := C_{sub}.\texttt{rights}\}) \quad} De$$

• **ShowRight.**

$$Do \xrightarrow{\quad CredShow(Cred_{right}, \{\texttt{revTag} \notin BL\ \wedge\ \texttt{rights} \supset NSet\}) \quad} V$$

• **RevokeRight.**

$$E \xrightarrow{\quad request\ revocation\ of\ the\ right\ with\ revocation\ tag\ revTag \quad} RM$$

RM. set $L = \{revTag\}$, while $L \neq \{\}$ do the following
1. remove value e from L, add e to blacklist BL
2. check archive for Sig^j_{rev} on tuple $(e, e^j_{1i}, \ldots, e^j_{ni}, encr^j)$
3. for each Sig^j_{rev} found do the following
   a. decrypt $encr^j$ and retrieve tuple $(e^j_{1u}, \ldots, e^j_{nu})$
   b. add values $e^j_k = e^j_{ku} + e^j_{ki}$ to L, for all $k \in \{1, \ldots, n\}$

Fig. 3. Credential-based implementation of the delegation model (2/2)

During the second phase, *De* sends a signature Sig_{de} to *I*. This signature includes tuple (Sig_{sub1}, Sig_{sub2}) and additionally proves that *De* is the owner of identifier id_{de} encoded into C_{de}. If Sig_{de} is accepted by *I*, and if (Sig_{sub1}, Sig_{sub2}) has not been shown to *I* before, *De* retrieves a new credential of which the attributes are based on the values encoded into C_{sub}.

ShowRight. During the *ShowRight* protocol, *U* shows her credential $Cred_{right}$ to *V*. Additionally, she proves that it contains sufficient rights for accessing *V*'s services and that it has not been revoked. The latter can be achieved using the efficient privacy-friendly blacklisting techniques of [9, 3].

RevokeRight. Revocation manager *RM* revokes a right by adding its revocation tag to a public blacklist *BL*. If *revTag* belongs to a main-right, all rights in its revocation tree are iteratively revoked by retrieving the signatures Sig^j_{rev} on *revTag* and by decrypting the encryptions $encr^j$. Note that RM will generally not be aware of the correct decryption key. In this case, decryption requires the interaction with a trusted third party.

Before performing a revocation, RM receives a revocation request from an entity E in the system. Requests from identified entities such as issuers or verifiers generally pose no problem, as they can easily be held accountable for their actions. Care must be taken, however, when requests are made by unidentified entities. These requests will only be granted if the requester can prove to be the owner of a credential $Cred_{rev}$ containing revocation tag *revTag.* The proof protocol is given in Figure 4. During the protocol, *Do* can either request the revocation of a sub-right issued by herself, or of a right owned by herself.

$$\begin{tikzcd}
Do \arrow[rrr, "Sig_{rev} = CredSign(Cred_{rev}, \{revTag \in \{\mathtt{e}, \mathtt{e}_1, \ldots, \mathtt{e}_n\}\}, revTag)"] & & & RM
\end{tikzcd}$$

Fig. 4. Revocation request for anonymous users

5 Evaluation

Anonymity and linkability requirements.

A1. Service access is anonymous and unlinkable, even if multiple entities collaborate and freely exchange their information. Since main-rights and sub-rights have an identical structure, right indistinguishability is also achieved.

A2. Provided that no revocations are performed, subright unlinkability is trivially achieved. When a right is revoked, all revocation tags of this right and of its sub-rights are retrieved and linked. A "skeleton" of the right's delegation tree can then be reconstructed. This skeleton contains as its nodes the revocation tags of possible sub-rights, but not the sub-rights themselves. Users who are willing to display the specifications of their revoked sub-rights, may place it at the correct position in the tree. As such, limited but nevertheless additional information concerning a user's delegation behavior may be retrieved.
One way to avoid these unwanted linkabilities is by not allowing any revocations. This is however not a reasonable solution. A good compromise is the adoption of "medium-size" validity periods. These time periods should be short enough to avoid most revocations on the one hand but long enough to avoid burdensome renewals on the other hand.

Security Requirements.

S1. All rights are unforgeable thanks to the unforgeability of credentials and the unforgeability of the *CredSign* signature scheme.

S2. Sub-rights are issued correctly. During the *DelegateRight* protocol, *Do* explicitly proves that the sub-right's validity periods and right specifications are more strict than or equal to what is specified in the main-right. Note that Do is not prohibited to issue revoked sub-rights. Issuing such rights would be useless, however, as they would be refused by V anyway.

S3. Transferability of rights can be discouraged by using non-transferable user secrets [2, 10]. For this, value id_u is constructed as a secret key or a credit card number. An exception to this adaptation is credential $Cred_{rev}$, which does not contain id_u. Here, transferring is discouraged by the fact that it may only harm its original owner Do. This is because (1) transferring $Cred_{rev}$ does not enable another user to employ its encoded revocation tags e_i, and (2) transferring $Cred_{rev}$ does enable other users to revoke the sub-rights issued by Do.

S4. When showing multiple rights to the same verifier. Consistency of these rights can be demonstrated by an additional proof that the `Owner` attribute is the same in all credentials.

S5. Rights are revocable and the revocation of a main-right implies the revocation of all the rights in its delegation tree. In addition, users cannot request the revocation of rights which are not issued or owned by themselves.

S6. Conditional deanonymization can easily be added using standard techniques [4]. During the *showRight* protocol, U simply provides V with a verifiable encryption of either her identity or of her right's revocation tag.

Extensions and adaptations.
In our construction, every right has a validity period and a set of right specifications. All types of sub-right can be issued, provided that their encoded constraints are a subset of what is specified in the main-right. In many applications, however, these system specifications are too limited. We now give some examples of extensions to the system. A detailed discussion on these and other extensions and on how to achieve them can be found in our technical report [8].

- By employing limited-show credentials, it is possible to limit the number of times that a right can be shown to a verifier. Note that in this case, the issuing of a sub-right which can be shown t times must imply the loss of t show instances for the main-right.
- The maximal depth of a right's revocation tree can be set to a fixed number. As an example, this depth may be set to 1 in the situation where a doctor may delegate sub-rights to her assistants, but where her assistants are not allowed to issue sub-rights of themselves.

Our system has the obvious drawback that I needs to be involved in every delegation. In applications with less strict privacy and functionality requirements, this dependability on I can be alleviated by a small transformation of the system [8]. First, we note that signature tuple (Sig_{sub1}, Sig_{sub2}) contains sufficient proof that De is entitled to a sub-right. Hence De can show her right by simply showing (Sig_{sub1}, Sig_{sub2}) and by proving some additional statements about the signed values. As an example, in order to prove that her right has not yet been revoked and that it is sufficient for accessing V's services. De can prove the predicates ($C_{sub}.\texttt{subRevTag} \notin BL$) and ($C_{sub}.\texttt{rights} \supseteq Nset$). Note that this procedure does not maintain the unlinkability of service access, the indistinguishabilty of rights or the delegation capability of the sub-right. If one of these features is needed by *De*, she gets back to the original protocol and contacts *I* for a credential.

Finally, our system can easily be extended to allow sub-rights which are created as a combination of rights situated in different main-rights.

6 Conclusion

This paper introduced a formal model for a delegation system and presented a credential-based implementation. The system provides both anonymous delegation for the individual as well as security for the organizations. A trade-off has been made between the security requirement of correct revocation and the anonymity requirement of the delegation process. It is an interesting problem to investigate whether this conflict can be solved, such that both revocation and sub-right unlinkability can be achieved.

References

1. Michael Backes, Jan Camenisch, and Dieter Sommer. Anonymous yet accountable access control. In *WPES*, pages 40–46, 2005.
2. S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy.* MIT Press, Cambridge, MA, USA, 2000.
3. Stefan Brands, Liesje Demuynck, and Bart De Decker. A practical system for globally revoking the unlinkable pseudonyms of unknown users. Technical Report CW472, Katholieke Universiteit Leuven, 2006.
4. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, pages 93–118, 2001.
5. Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.
6. Jan Camenisch, Dieter Sommer, and Roger Zimmermann. a general certification framework with applications to privacy-enhancing certificate infrastructures. Tech. Rep. RZ 3629, IBM Zurich Research Laboratory, July 2005.
7. Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT*, pages 125–142, 2002.
8. Liesje Demuynck and Bart De Decker. Credential-based systems for the anonymous delegation of rights. Technical Report CW468, K.U. Leuven, 2006.
9. Liesje Demuynck and Bart De Decker. How to prove list membership in logarithmic time. Technical Report CW470, Katholieke Universiteit Leuven, 2006.
10. Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, pages 184–199, 1999.
11. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.
12. Sven Wohlgemuth and Günter Müller. Privacy with delegation of rights by identity management. In Günter Müller, editor, *ETRICS*, volume 3995 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2006.