

Credential-based systems for the anonymous delegation of rights

Liesje Demuynck
Bart De Decker

Report CW468, November 2006



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Credential-based systems for the anonymous delegation of rights

Liesje Demuyneck
Bart De Decker

Report CW468, November 2006

Department of Computer Science, K.U.Leuven

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 three implementations. Our implementations are based on credential systems and provide both anonymity for the individual and security for the organizations. The implementations are compared based on their functionality, privacy and security characteristics. Additionally, some guidelines are given for choosing a particular implementation based on the application's requirements.

CR Subject Classification : E.3 [Data]: Data Encryption – Public key cryptosystems.

Credential-based systems for the anonymous delegation of rights

Liesje Demuynck* Bart De Decker

1 Introduction

The concept of authentication and authorization has long been studied in computer science. Intuitively, all proposed 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 [3, 5].

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 the access to some specific files in the database. She will prefer this delegation procedure to be anonymous, such that no central authority can monitor her delegation behavior. On the other hand, measures must be taken to control her delegation capabilities.

This paper introduces a general delegation model and presents three implementations. Our implementations are based on credential systems and provide both anonymity for the individual and security for the organizations. The implementations are compared based on their functionality, privacy and security characteristics. Additionally, some guidelines are given for choosing a particular implementation based on the application's requirements.

The outline of this paper is as follows. Section 2 presents a formal model of the delegation system. Section 3 then describes the basic building blocks used in our implementations: commitment schemes, credential systems and verifiable encryptions. Our credential-based implementations are presented and evaluated in Section 4. Finally, Section 5 provides a comparison of the implementations and gives some general guidelines.

2 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.

*Research Assistant of the Research Foundation - Flanders (FWO - Vlaanderen)

$E \leftrightarrow RG$:	<i>Registration</i> (certificates)
$I \leftrightarrow U$:	<i>IssueRight</i> (RSpecs) returns IssueTrans; R
$Do \leftrightarrow De$:	<i>DelegateRight</i> ([I], MR, SRSpecs) returns IssueTrans; SR
$U \leftrightarrow V$:	<i>ShowRight</i> (R, showProperties)
RM	:	<i>RevokeRight</i> (revTag)

Table 1: general delegation model - protocol overview.

2.1 Roles and protocols

Roles. An entity E in the system is either a *system registrar* RG , an *organization* O or a *user* U . Users may perform their interactions anonymously, while the system registrar and organizations must at all times be identifiable. The system registrar registers both users and organizations to the system. She also provides them with the necessary certificates for using or providing the system’s services.

An organization O is either an *issuer* I , a *verifier* V or a *revocation manager* RM . An issuer issues rights to the users in the system. Such a right contains a set of specifications and a validity period, and can be shown to a verifier a fixed maximal number of times. Additionally, it can be used to issue sub-rights, which in turn can 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.

A user U 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 right as its sub-right. Note that a right can be both a main-right with respect to 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. Double-headed arrows represent interactive protocol.

An entity registering to the system performs the *Registration* protocol with RG . During this protocol, she provides RG with the necessary external certificates for accessing the service. Note that the registration of an organization typically consists of the certification of her key material. Therefore, in the remainder of this paper, we will focus on user registration only.

A registered user U may retrieve a right R satisfying specifications $RSpecs$ by performing an *IssueRight* protocol with I . As a result, I retrieves a transcript $IssueTrans$.

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 (i.e. users, organizations and system registrar) can freely exchange their information. In particular, users may provide 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, credit card numbers and revocation or tracking information of sub-rights issued by themselves.

2.3 Requirements

The system's requirements are divided into functionality, anonymity, security and performance requirements. Apart from the security requirements, all requirements are optional.

Functionality requirements.

- F1. *Ad-hoc delegation*. Ad-hoc delegation gives a delegator the power to decide on the spot which rights to pass on to which entities. Although not mandatory, ad-hoc delegation greatly improves the flexibility of delegation.
- F2. *Expressiveness of constraints*. The creator of a right may enforce some constraints on this right. We distinguish between constraints on usage and constraints on delegation.
 - (a) *Constraints on usage*. These constraints can be applied to any right and limit the general usability of this right. Examples of such constraints include the specification of validity periods or a limit on the number of show-protocols for the right.
 - (b) *Constraints on delegation*. These constraints can be applied to any right and limit the usability of the right for issuing corresponding sub-rights. The following constraints may be set:
 - i. *A limit on the number of sub-rights that can be issued based on a right*.
 - ii. *A limit on the depth of a right's delegation tree*. As an example, this limit is set to one in situations where a doctor may issue sub-rights to her assistants, but where her assistants are not allowed to issue sub-rights of themselves.
 - iii. *Constraints on the types of sub-rights that can be issued*. The constraints on a sub-right must naturally be more strict or at least as strict as the constraints on its main-right. In particular, the sub-right's right specifications, number of usages and validity period must be less than or equal to what is specified for the corresponding

main-right. In addition, the sub-right’s delegation tree depth and number of sub-rights must be strictly less than what is specified for the main-right.

Next to these traditional constraints, additional constraints may be set by the issuer of the main-right. For example, a main-right allowing full access to a database, may only be allowed to issue sub-rights for accessing a very limited subpart of this database.

- iv. *Constraints on the choice of sub-right owners.* Many applications require constraints on the characteristics of potential sub-right owners. A main-right enabling doctor’s access to a medical database, for example, may only be used to issue sub-rights to the assistants of this particular doctor.

Anonymity 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 right was obtained: i.e. whether it was obtained through an *IssueRight* protocol or through a *DelegateRight* protocol.
- 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 the other entities.
- A3. *Monitoring of rights.* In some applications, the issuer of a right wants to monitor the usage of this right. She then receives additional information about when a right has been shown and to which verifier. Monitoring should only be allowed for the rightful issuer and may not reveal any additional information on the identity of the right’s owner. Whenever a monitorable right has issued sub-rights, these sub-rights must also be monitorable by the issuer of the original right. We distinguish between *immediate monitoring* (i.e. the right can be monitored right from the start) and *delayed monitoring* (i.e. monitoring is delayed until after the occurrence of a predefined event). Note that the presence of delayed monitoring implicitly implies the possibility for immediate monitoring. This is achieved by setting the predefined event to “null”.

Security requirements. The following security requirements are mandatory. If one of them is missing, the entire system is insecure.

- S1. *Unforgeability.* This requirement states the inability of users to successfully show a right which was not obtained by means of an *IssueRight* or of a *DelegateRight* protocol.

- S2. *Correct sub-rights.* Users should not be able to successfully show a sub-right of which the encoded rights are less strict than what is specified in its main-right.
- S3. *Non-transferability.* Unless explicitly specified otherwise, the legitimate owner of a right must not be able to pass on her “actual” right to other users. Here, the “actual” right refers to the digital tokens constituting the right. Hence, the situation of passing on a right by issuing a sub-right identical to the original right is, if allowed by the right, still possible.
- S4. *Consistency of rights.* Unless explicitly specified otherwise, rights must be bound to their owner. In particular, it may not be possible for users 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 tree. In addition, users should be prohibited to request the revocation of rights they did not issue themselves.
- S6. *Conditional deanonymization.* In case of abuse of a right, appropriate measures should be taken. We distinguish two types of actions.
 - (a) conditional retrieval of the owner’s identity.
 - (b) conditional retrieval of the right’s issue transcript, enabling the right’s revocation.

Performance requirements. The adopted protocols must be efficient, such that practical implementations are possible. Furthermore, additional entities for the delegation should be prohibited as much as possible. Indeed, an ideal *DelegateRight* protocol is a protocol between a main-right owner and a sub-right owner only. Whenever an additional entity is needed, this entity must be available constantly. This implies a loss in efficiency, and hence this scenario should be avoided as much as possible.

3 Protocols in our base system

Our constructions are based on commitments, credential systems and verifiable encryptions. We briefly introduce these concepts and their primitives. All communication between entities must be performed over anonymous communication channels.

Commitments. A commitment [13, 10] can be seen as the digital analogue of a “non-transparent sealed envelope” [12]. 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

$$E : Comm, OpenInfo = Comm(\{\mathbf{attrName} := attrValue, \dots\})$$

enables an entity E to create a commitment $Comm$ to 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.

$$E_1 \rightarrow E_2 : CommProps(Comm, P(\mathbf{attr1}, \dots))$$

The public input to this protocol is both a commitment $Comm$ and a boolean predicate P concerning $Comm$'s attributes. 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 about $Comm$ and its attributes.

The $CommProps$ protocol is an interactive protocol. Nevertheless, we adopt the one-headed arrow $E_1 \rightarrow E_2$ in our notation. This is to avoid any ambiguity that may result from our traditional (double-headed) arrow notation. Intuitively, the one-headed arrow makes clear that it is E_1 who proves commitment properties to E_2 , and not the other way around.

Credentials. A credential system [3, 5] allows for anonymous yet accountable transactions between users and organizations. In the remainder, we employ the system proposed by Camenisch et al. [5, 2, 9]. The system's $CredGet$ and $CredShow$ protocols are interactive protocols. Again, we adopt one-headed arrows to avoid any confusion in their interpretation.

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

$$U \leftarrow I : Cred = CredGet((sl_1, \dots, sl_n), \{\mathbf{attr}_1 := G(\cdot), \dots\})$$

$Cred$ contains a set of attributes and a set of show-limits (sl_1, \dots, sl_n) corresponding to different show-modes. Additionally, it contains a secret key for showing it to a verifier.

Each attribute is constructed as a separate function $G(\cdot)$ of public values and attributes encoded into previously shown credentials or commitments. As an example, \mathbf{attr}_1 may be constructed as $\mathbf{attr}_1 := Cred_x.a_1 + 5$, where $Cred_x.a_1$ refers to attribute a_1 of a previously shown credential $Cred_x$. I cannot find out any information concerning the credential's final attributes, apart from the fact that they are constructed correctly based on $G(\cdot)$.

A show-limit $sl_i \in \mathbb{N} \cup \{*\}$ ($i \in \{1, \dots, n\}$) denotes the maximal number of times that a credential can be shown in show-mode i . Here, notation "*" denotes a show-limit which is set to infinity. Whenever a limit sl_i is exceeded, this is detected and appropriate actions are taken. In the remainder, we make abstraction of the available techniques [7] for including these show-limits. Instead, we assume that a credential can only be shown k_i times in show-mode i , and that showing it a $(k_i + 1)$ -th time will simply not succeed.

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

$$U \rightarrow V : CredShow(Cred, (t_1, \dots, t_n), P(\mathbf{attr}_1, \dots))$$

As a result, U loses $t_i \geq 0$ show-instances for each show-mode i ($i \in \{1, \dots, n\}$). Note that t_i may also be zero, which can be seen as a temporal suspension of the credential's limited-show property.

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 $(\mathbf{attr}_1 > C_x.a_1 \wedge \mathbf{attr}_1 < C_x.a_2)$, where $C_x.a_1$ and $C_x.a_2$ refer to attributes a_1 and 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.

An eavesdropper listening in on the execution of either a $CredGet$ or a $CredShow$ protocol, cannot find out any other information than what can be deduced by the organization involved in the

protocol. Hence, as long as the interaction is performed over an anonymous communication channel and as long as no confidential information is communicated between the parties, no confidentiality protection is needed on the communication lines.

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

$$U : Sig = CredSign(Cred, (t_1, \dots, t_n), P(\mathbf{attr}_1, \dots), msg)$$

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

Verifiable encryptions. Verifiable encryptions [1, 6, 8] have all the characteristics of regular encryptions. Additionally, they enable their creator U to demonstrate properties of the encrypted plaintext. As an example, U can prove to E that the encrypted plaintext is encoded as an attribute in a credential or commitment. This will be denoted as a predicate $c = VE(\mathbf{x})$, which can be included in the *CommProps* or *CredShow* primitives. Here, c refers to the ciphertext and \mathbf{x} refers to the credential's (or commitment's) attribute \mathbf{x} .

Note that c is created using a public key pk of which the corresponding secret key sk may not be known by E . 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 (*S1*) and allow for service accesses which are anonymous (*A1a*) and unlinkable (*A1b*). By combining them with commitments and verifiable encryptions, additional properties such as non-transferability (*S3*), consistency of credentials (*S4*), conditional deanonymization (*S6*) and revocation (*S5*) can easily be added. We now give a hint on how these extensions can be achieved [3, 5].

1. Non-transferability can be achieved by including a personal secret s_u as an attribute into the credential. This may, for example, be a credit card number or the key to a bank account. Its value is unknown to I and is never explicitly shown to V . As the transfer of a right implies the transfer of this secret, users will not be inclined to give away their rights.
2. Consistency of rights is obtained by encoding a unique identifier into all credentials belonging to the same user. When multiple rights are shown to V , U additionally demonstrates that their encoded identifiers are the same. Note that this identifier is not known to I and that it is never shown to V . Personal secret s_u can be used for this purpose.
3. Revocation can be achieved by encoding a unique issuer-defined revocation tag e into the credential. The right is then revoked by adding e to a public blacklist BL . Whenever U shows her credential to a verifier, she additionally proves that her revocation attribute is not included in BL .
4. Conditional deanonymization requires two additional attributes encoded into the right. The first attribute id is unknown to I and uniquely identifies the right's owner. The second attribute e is defined by I and uniquely specifies the right's issue protocol (this might, for

example, be the right’s revocation tag). Whenever the right is shown to a verifier, U provides V with verifiable encryptions $VE(id)$ or $VE(e)$. Upon fulfillment of the deanonymization condition, these value will be decrypted. Decryption of $VE(id)$ enables a link with the right’s owner, while $VE(e)$ ’s decryption enables the retrieval of the right’s issue transcript.

4 Credential-based implementations

We now present our credential-based techniques for implementing the general delegation model of Section 2. Section 4.1 presents a simple but limited technique based on transferable credentials. Section 4.2 demonstrates a technique for issuing sub-rights based on signatures and Section 4.3 gives an extension of this technique to credentials. Finally, Section 4.4 briefly introduces a hybrid system based on the ideas of Sections 4.2 and 4.3.

4.1 Transferable credentials

This first technique employs transferable credentials and pseudonyms to construct rights that can be further delegated. Section 4.1.1 gives a global overview of the system and its primitives. The system itself is described in Section 4.1.2 and evaluated in Section 4.1.3.

4.1.1 Setting

Each user in the system has a non-sharable external user secret s_u (for example, a credit card number or a secret signing key). We assume the value of this secret not to be in the control of the user. Instead, the value is chosen randomly by a trusted authority or by trusted hardware. Based on s_u , U can create pseudonyms P_u having some very useful properties. In particular, P_u unconditionally hides its underlying value s_u . Next to this, only the legitimate creator of P_u can prove to be the owner of the pseudonym.

A main-right is now represented as a transferable credential (i.e. a credential which does not explicitly encode s_u) containing a number of pseudonyms. Each pseudonym belongs to a particular entity and is associated with a specific sub-right. The right is delegated by passing on the credential. The resulting sub-right cannot be further delegated, but can be shown to a verifier by showing the credential and by additionally proving ownership of a pseudonym encoded into the credential. We now describe the cryptographic constructs to achieve this system.

User secret s_u is a random number in \mathbb{Z}_n^* with n an RSA modulus. For example, s_u may be chosen randomly by a trusted authority and may be used as a pre-image for the user’s credit card number. Furthermore, two functions f and nym are defined as follows.

$$\begin{aligned} f : \mathbb{Z}_n^* &\rightarrow \mathbb{Z}_n^* : x \mapsto x^v \pmod n & \text{and} \\ nym : \mathbb{Z}_n^* \times \mathbb{Z}_n^* &\rightarrow \mathbb{Z}_n^* : (s, r) \mapsto s \cdot f(r) \pmod n \end{aligned} \quad (1)$$

Value v is a prime number such that $\gcd(v, \phi(n)) = 1$. This value should be chosen very carefully to enable efficient zero-knowledge proofs of the relation $y = f(x)$ for values y and x encoded into a commitment or credential. Bresson and Stern [4] provide a proof technique requiring a complexity of $O(\log_2(v))$. As such, the popular value $v = 2^{16} + 1$ may be used as an exponent.

Note that f is a one-way function. Function nym can be used to compute pseudonyms $P_u = nym(s_u, r)$, where r is a random value in \mathbb{Z}_n^* . User U can easily prove to be the owner of the pseudonym, by proving knowledge of values s_u and r , such that s_u is the user's non-transferable user secret and $P_u = nym(s_u, r)$. The following properties can be shown to hold.

Proposition 1. P_u unconditionally hides s_u

Proof. For every value $s_u \in \mathbb{Z}_n^*$, a unique value $r = (P_u/s_u)^{1/v} \pmod n$ can be computed such that $P_u = s_u \cdot r^v \pmod n$. As a consequence, P_u reveals nothing about its underlying value s_u . \square

Proposition 2. Under the RSA assumption for a public RSA key (n, v) , the following actions cannot be achieved.

1. Given a tuple $(P, s_u) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$, find a value r such that $P = nym(s_u, r)$.
2. Given a tuple $(s_u, s_{u'}) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$, find values r and s such that $nym(s_u, r) = nym(s_{u'}, s)$.

We only give a brief sketch of the proofs for both statements.

1. Let \mathcal{A} be an adversary who receives as input a tuple $(s_u, P) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$ and who subsequently outputs with non-negligible probability a value r such that $P = s_u r^v \pmod n$. We now sketch how an adversary \mathcal{B} can use \mathcal{A} to break the RSA assumption. In a first step, \mathcal{B} retrieves as input a random value $e \in \mathbb{Z}_n^*$. She picks $s_u \in_{\mathcal{R}} \mathbb{Z}_n^*$, computes $P = e \cdot s_u \pmod n$ and feeds (s_u, P) as input to \mathcal{A} . When receiving an answer r from \mathcal{A} , \mathcal{B} checks whether $P = s_u r^v \pmod n$. If so, she outputs r . As $r^v \equiv P/s_u \equiv e \pmod n$, \mathcal{B} has now broken the RSA assumption.

2. Let \mathcal{A} be an adversary who receives as input a tuple $(s_u, s_{u'}) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^*$ and who subsequently outputs with non-negligible probability a tuple (r, s) such that $nym(s_u, r) = nym(s_{u'}, s)$. We now sketch an algorithm \mathcal{B} that can be used to break the RSA assumption. \mathcal{B} receives as input a random value $e \in \mathbb{Z}_n^*$. She chooses $s_u \in_{\mathcal{R}} \mathbb{Z}_n^*$, computes $s_{u'} = s_u/e \pmod n$ and feeds tuple $(s_u, s_{u'})$ as input to \mathcal{A} . After a while, she receives tuple (r, s) from \mathcal{A} . She checks whether $nym(s_u, r) = nym(s_{u'}, s)$. If so, she outputs $x = s/r$. As $x^v \equiv (s/r)^v \equiv s_u/s_{u'} = e \pmod n$, \mathcal{B} has broken our RSA assumption.

This concludes our sketch of the proof.

Note that the properties of Proposition 2 can be seen as a special type of one-wayness and collision-resistance, which are achieved when the first argument s_u of function nym is a random predefined value.

Corollary 1. Only the legitimate creator U of a pseudonym P_u can find values s_u and r such that $P_u = nym(s_u, r)$ and such that s_u is U 's non-transferable usersecret.

4.1.2 The delegation system

The system is depicted in Figure 1. Table 2 gives an overview of the communication channels used during the protocol. As to protect the anonymity of users, most communication is performed over anonymous communication channels. Authentication, integrity and confidentiality can be realized by running TLS/SSL over anonymous communication channels. We now give a brief overview of the protocols.

Figure 1: shared credential-based implementation of the delegation model

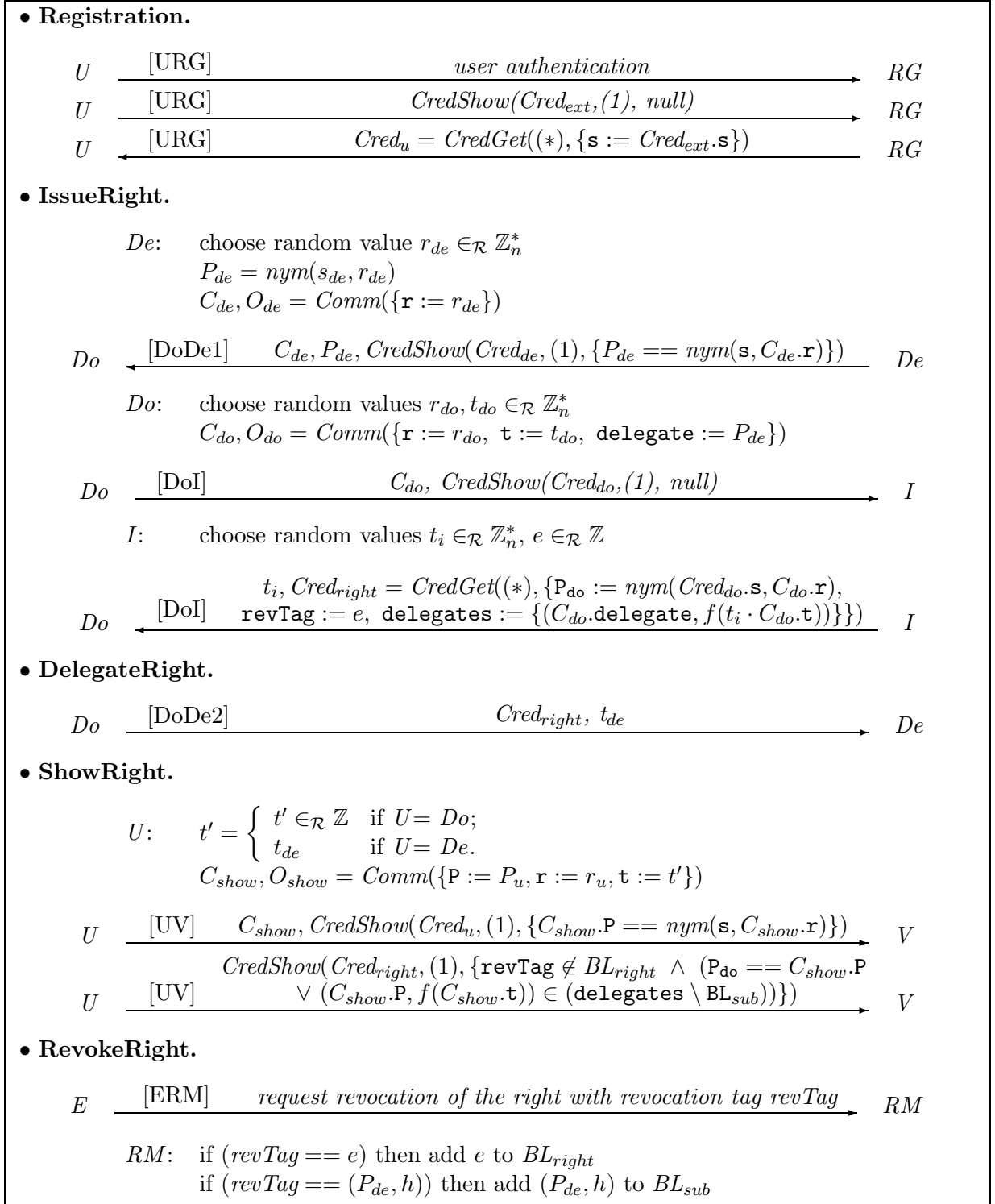


Table 2: Communication channels for the shared credential-based implementation

channel	authentication	integr.	confid.	anon.
[URG]	RG to U U to RG	yes	no	no
[DoDe1]	optional: De to Do	yes	yes	yes
[DoDe2]	optional: De to Do	yes	yes	yes
[DoI]	I to Do	yes	no	yes
[UV]	V to U	yes	no	yes
[ERM]	RM to E optional: E to RM	yes	no	yes

Registration. A user entering the system must be in possession of an external credential $Cred_{ext}$, containing as its attribute a non-transferable and randomly chosen user secret s_u . U shows $Cred_{ext}$ to RG and optionally identifies herself. If this is successful, she receives a new credential $Cred_u$ containing her user secret s_u . In the remainder, we will refer to this credential as $Cred_{do}$, $Cred_{de}$ or $Cred_u$, depending on the owner’s role as respectively a delegator, a delegate or a user.

IssueRight. For ease of representation, we describe the issuing protocol for rights encoding only one potential delegate and limited to issuing sub-rights identical to the original right. Generalizations towards multiple delegates and sub-right types can straightforwardly be achieved by adding additional attributes.

The protocol consists of two phases, which can be separated in time. During the first phase, De sends Do her pseudonym P_{de} . She additionally proves that it is correctly formed as $P_{de} = nym(s_{de}, r_{de})$, where s_{de} is De ’s user secret and r_{de} is a random value. Depending on the situation, Do may know De ’s identity or both parties may be anonymous with respect to each other. Furthermore, as to hide P_{de} from possible eavesdroppers, the communication channel [DoDe1] is confidentiality-protected.

During the second phase, Do contacts I to retrieve the actual right. This right is represented as an unlimited-show credential containing three attributes; the delegator’s pseudonym P_{do} , a revocation tag **revTag** chosen by the issuer and a list **delegates** of potential delegates. Each potential delegate is represented as a tuple (P_{de}, h) with P_{de} a pseudonym for De and $h = f(t_{de})$ for $t_{de} = t_{do} \cdot t_i \bmod n$. Value h will be used both for revocation purposes and for delegation purposes. Its value is chosen jointly by Do and by I and has some interesting properties:

- Do cannot manipulate h ’s value. As will become clear later, this prevents users for requesting illegal revocations.
- Only Do knows pre-image $t_{de} = f^{-1}(h)$. This ensures that only Do can issue a sub-right to De .

Note that the communication channel [DoI] between Do and I does not need to be confidentiality protected. In particular, value t_{de} cannot be deduced from listening in on the communication. Also, even though an eavesdropper may obtain the right’s revocation tag e , she could also obtain

this information directly from issuer I based on our assumption that all entities in the system may freely exchange their information (Section 2.2).

DelegateRight. Delegation of a right is very cheap. It is achieved by sending $Cred_{right}$ and $t_{de} = t_i \cdot t_{do} \bmod n$ to De . As both values are confidential information, the communication is performed over a confidentiality-protected communication channel. The retrieved sub-right can not be further delegated and can be used multiple times by De in interactions with V .

ShowRight. U first constructs a commitment C_{show} . This commitment contains her pseudonym P , value r used during the creation of P , and a value \mathbf{t} which is set to $f^{-1}(h)$ if the user is a delegate. She then shows both $Cred_u$ and $Cred_{right}$ to V , and additionally proves the following properties.

1. C_{show} 's attribute P is correctly formed based on her user secret and random value r .
2. Right $Cred_{right}$ is not revoked.
3. Pseudonym P equals either the pseudonym P_{do} of the $Cred_{right}$'s original owner (i.e. Do), or it can be used with value \mathbf{t} to construct a tuple $(P, f(\mathbf{t}))$ representing a sub-right which has not been revoked.

Note that only a legitimate owner of an unrevoked right can successfully prove these statements.

RevokeRight. A main-right is revoked by adding its revocation tag e to blacklist BL_{right} . A sub-right is revoked by adding its identifying tuple (P_{de}, h) to blacklist BL_{sub} .

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. In general, these requests will only be granted if the requester can prove to be the owner of a main-right containing the tuple (P, h) requested for revocation. This proof protocol is given in Figure 2. An important condition for this technique to work, is that h 's value cannot be manipulated by Do during the *IssueRight* protocol. If h could be manipulated by Do , a malicious user could successfully invoke the revocation of a tuple (P, h) . First, she performs an *IssueRight* protocol for a credential $Cred_{right}$ of which the **delegate** attribute is set to $\{(P, h)\}$. Afterwards, she requests the revocation of (P, h) by performing the protocol of Figure 2.

4.1.3 Evaluation

The system presented above is limited to rights issued by I and to direct sub-rights thereof. Hence, the depth of a right's delegation tree is at most 1. In this evaluation, we will refer to a right issued by I as a main-right, and to all other rights in its delegation tree as its sub-rights.

Functionality requirements.

- F1. True ad-hoc delegation is not possible. During the *IssueRight* protocol, Do specifies a list of potential delegates. Users not specified in this list will never be able to retrieve a valid corresponding sub-right. Users specified in the list may or may not retrieve a corresponding sub-right. This depends on the choice of the main-right owner.

these pseudonyms to real identities, however, unless these pseudonymous users come forward themselves.

A3. Delayed monitoring of rights can be achieved using the verifiable pseudorandom function proposed by Dodis and Yampolskiy [11]. This function has a form $F(w, x) = g^{1/(w+x+1)}$, for secret key w and input x , both elements of \mathbb{Z}_q , and for a generator g of a group G_q of prime order q in which the decisional Diffie-Hellman inversion problem is hard. Given public output y the relation $y = F(w, x)$ for a commitment's or credential's attributes w and x can easily be proven in zero knowledge. Delayed monitoring of main-rights (and hence also of its corresponding sub-rights) is then achieved as follows.

- Adaptations to the *IssueRight* protocol
 - Before the *IssueRight* protocol, *Do* constructs a pseudonym $P_{do} = nym(s_{do}, r_{do})$ that will be encoded in the final right. She sends this value to *De* and proves, using the *CredShow* protocol for $Cred_{do}$, that this value is constructed correctly.
 - During the first phase of the *IssueRight* protocol, *De* chooses a random value v and includes it as an additional attribute into C_{de} . She also computes $y = f(v)$ and $V_v = VE(v)$ and transforms the original *CredShow* protocol for $Cred_{de}$ into the following *CredSign* protocol: $Sig_{de} = CredSign(Cred_{de}, (1), (P_{de} == nym(\mathbf{s}, C_{de}.\mathbf{r}) \wedge y = f(C_{de}.\mathbf{v}) \wedge V_v = VE(C_{de}.\mathbf{v})), (P_{do}, C_{de}, P_{de}, y, V_v))$. Signature Sig_{de} is then sent to *Do*.
 - During phase two of the *IssueRight* protocol, *Do* picks a random value w and encodes it as an additional attribute into C_{do} . She computes $z = f(w)$ and $V_w = VE(w)$, and sends Sig_{de} , z and V_w to *I*. By means of the *CredShow* protocol for $Cred_{do}$, *Do* then proves to be the owner of pseudonym P_{do} in signature Sig_{de} . She also proves her knowledge of value w underlying the construction of z and V_w . This is done by means of the *CommProps* primitive for C_{de} .
 - Finally, *Do* receives a credential $Cred_{right}$ containing an additional attribute $\mathbf{z} := z$ and a **delegates** attribute containing a tuple of the form (P_{de}, h, y) .
- Adaptations to the *ShowRight* protocol.
 - During the *ShowRight* protocol for a main-right, *Do* computes $d = F(w, i)$ for $i \in D$ with D a predefined domain. Note that each value i should be used at most once. *Do* then sends d to *V* and proves that it is constructed correctly based on the value $f^{-1}(Cred_{right}.\mathbf{z})$. If the *ShowRight* protocol is successful, *V* publishes d .
 - During the *ShowRight* protocol for a sub-right the same steps are executed, this time using values v instead of w and $Cred_{right}.\mathbf{delegates}.\mathbf{y}$ instead of $Cred_{right}.\mathbf{z}$.
 - Note that right indistinguishability can still be maintained by using appropriate OR-connectives.
- Upon fulfillment of the tracking condition, *I* requests the decryptions w of V_w and v of V_v and then computes all values $F(w, i)$, $F(v, i)$ for $i \in D$. She can now link all actions performed by the right or by any of its sub-rights. Note that in order to prevent unauthorized tracking by malicious entities, *I* must authenticate before retrieving the decryptions.

Delayed monitoring of only sub-rights can be achieved by performing some minor adjustments on the protocol as described above.

- During the *IssueRight* protocol, all actions concerning w , z and V_w are skipped.
- The *ShowRight* protocol of a main-right is the same as in Figure 1. Note again that right indistinguishability can be maintained by the adoption of OR-connectives.
- Only the rightful owner of the main-right can successfully request the decryption of an encryption V_v included into Sig_{de} . For this, she proves to be the owner of the signed pseudonym P_{do} which is also included in Sig_{de} .

Immediate monitoring can readily be applied from delayed monitoring by setting the tracking condition to **always**.

Security requirements.

- S1. Unforgeability of main-rights is guaranteed by the unforgeability of digital credentials. Sub-rights cannot be forged due to the unforgeability of credentials, Corollary 1 and the one-wayness of f .
- S2. Sub-rights are correct. All constraints on a sub-right are explicitly encoded into credential $Cred_{right}$. As this credential is shown as part of each *ShowRight* protocol, it is impossible for users to successfully show an incorrectly constructed sub-right.
- S3/4. Non-transferability and consistency of rights is achieved using similar ideas as described in Section 3. A slight difference with this solution is that here, the non-transferable user secret s_u is not explicitly encoded into the right. Instead it is used as a basis for constructing the encoded pseudonyms.
- S5. All rights are revocable and the revocation of a main-right implies the revocation of all the sub-rights in its revocation 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 similar techniques as described in Section 3. During the show protocol, the user provides V with a verifiable encryption of her identity or of her right's revocation tag. Note that the former can be achieved by encoding U 's identity into credential $Cred_u$.

Performance requirements. The issuing of sub-rights is very cheap and does not require the help of any additional entity. The complexity of the system itself, however, rises with every additional delegate. Indeed, as $Cred_{right}$ contains a dataset for each delegate, its structure becomes larger if more delegates are encoded. Furthermore, Do must perform phase one of the *IssueRight* protocol for each of the delegates.

We note that the *ShowRight* protocol can be optimized if rights indistinguishability is not required. The proven statement is then split up into different statements for main-rights and sub-rights. The resulting protocols are a little less expensive than the current protocol.

Table 3: Communication channels for the signed warrant-based implementation

channel	authentication	integr.	confid.	anon.
[URG]	RG to U U to RG	yes	no	no
[DoI]	I to Do	yes	no	yes
[DoRM]	RM to Do	yes	no	yes
[DoDe]	optional: De to Do	yes	yes	yes
[DoV]	V to Do	yes	no	yes
[DeV]	V to De	yes	no	yes
[ERM]	RM to E optional: E to RM	yes	no	yes

4.2 Signed warrants

Next, we describe an implementation of the delegation model based on signed warrants. A description of the system and its protocols is given in Section 4.2.1. It is evaluated in Section 4.2.2.

4.2.1 Delegation system

As in Section 4.1, this implementation is limited to rights issued by I (henceforth called main-rights) and to direct sub-rights thereof. Main-rights are represented as digital credentials, while sub-rights are represented as signatures. The attributes of the main-right’s credential consist of a tuple $(s_u, e, RSet)$, where s_u is the owner’s user secret, e is the right’s revocation tag and $RSet$ is the specification of the right. A sub-right is issued by constructing a new tuple $(s'_u, e', RSet')$ and by signing a commitment on this tuple. This signature is created with the main-right’s credential by using the *CredSign* protocol. Note that we sign a commitment rather than the actual tuple $(s'_u, e', RSet')$. This way the tuple is hidden from any third parties. A sub-right can then be shown to a verifier by presenting her with the signature and by proving some claims about the signed commitment. Different show-protocols based on the same sub-right can be linked to each other.

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 this protocol, Do retrieves a credential $Cred_{rev}$ containing a list of revocation tags. RM does not know the final value 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 also occurring in $Cred_{rev}$, $Cred_{rev}$ is invisibly bound to the main right’s credential.

The system is depicted in Figures 3 and 4. Table 3 gives an overview of the communication channels and their characteristics. We now give a brief overview of the protocols.

Registration. Registration is identical as in Section 4.1.

IssueRight. Do first proves to be registered to the system. If successful, she retrieves a k -show credential $Cred_{right}$ containing three attributes: a copy of attribute s_{do} in $Cred_{do}$, an issuer-chosen revocation tag e and a specification $RSet$ of rights and validity periods.

Figure 3: signed warrant-based implementation of the delegation model - registration and main-right protocols

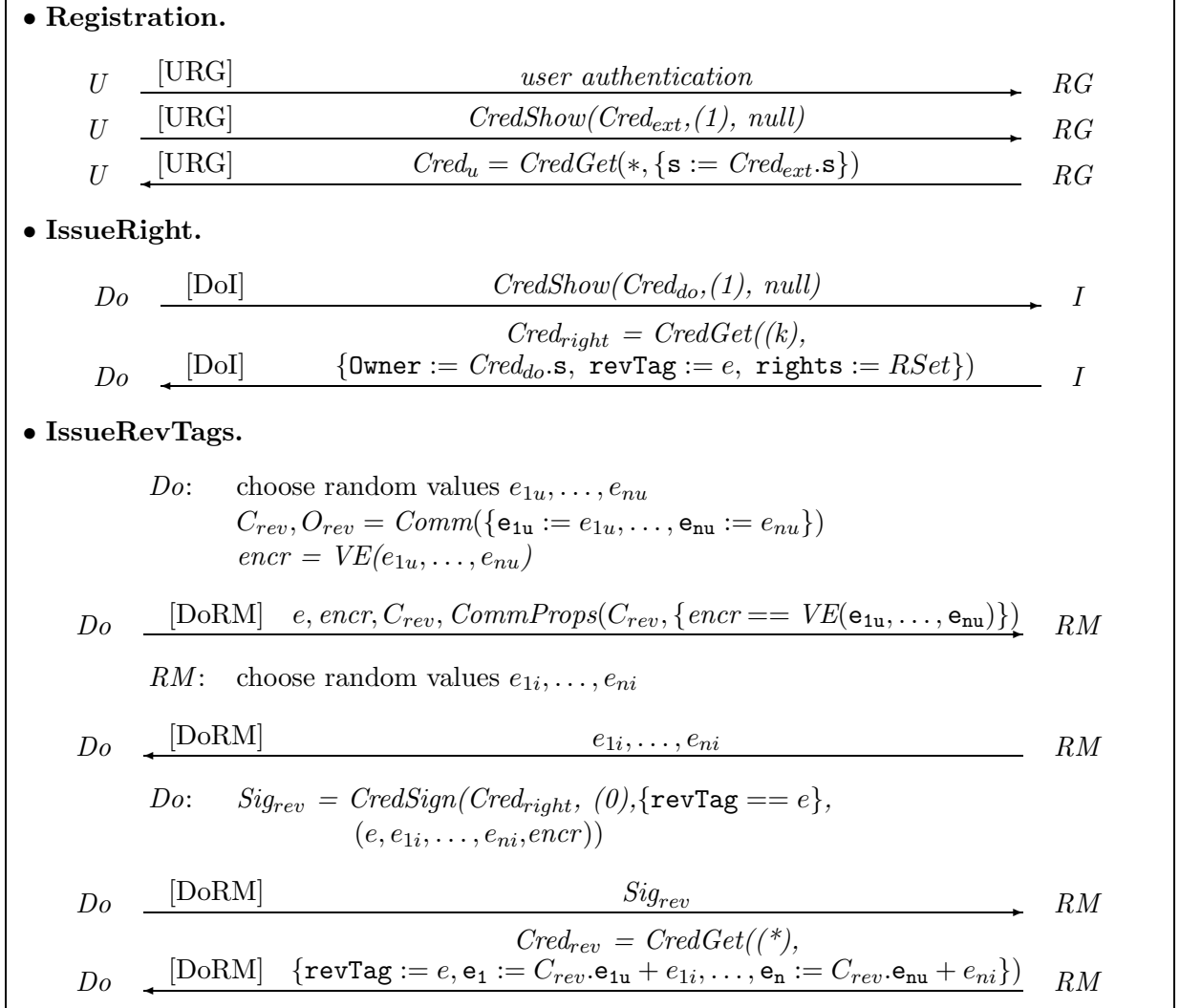
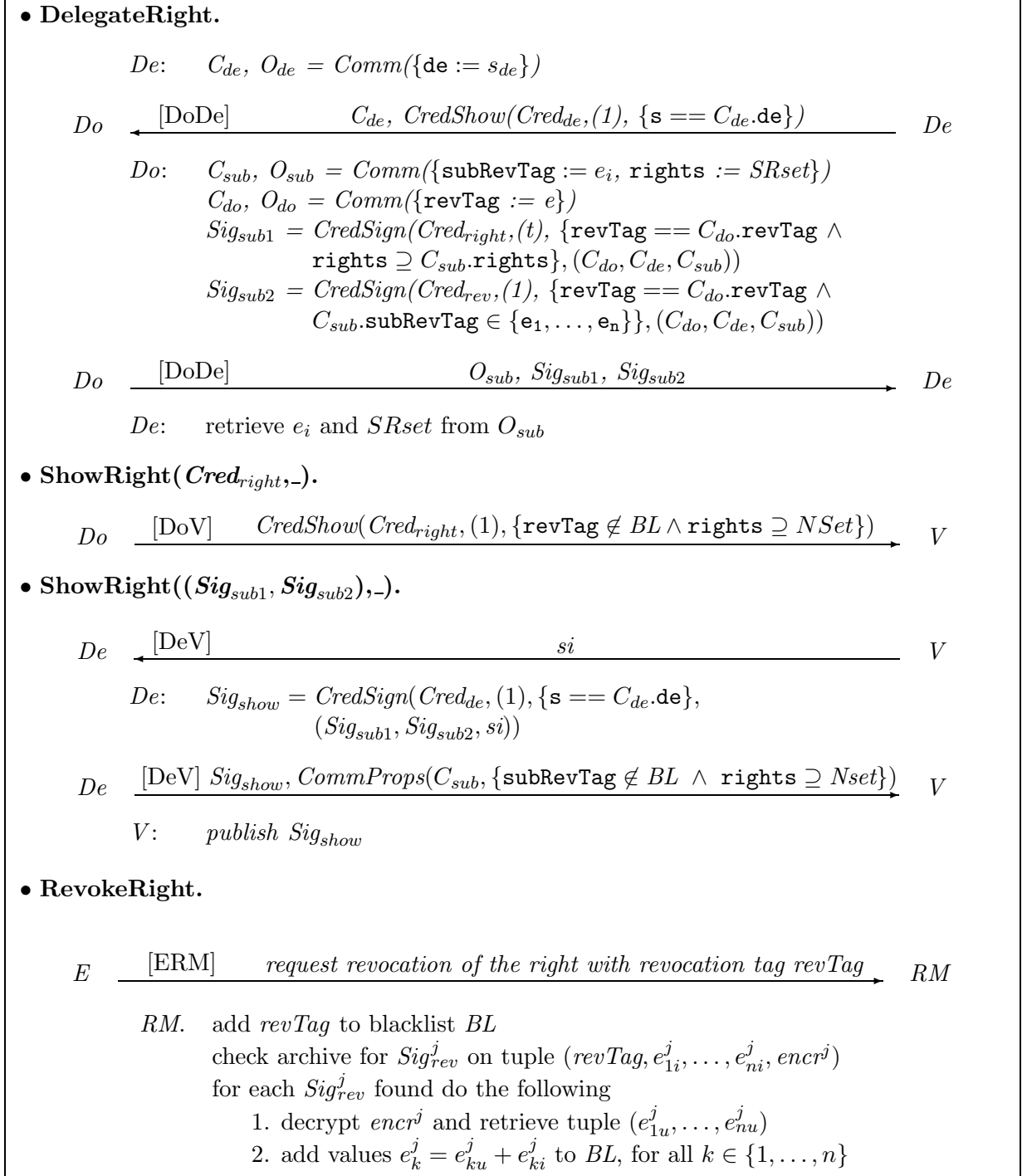


Figure 4: signed warrant-based implementation of the delegation model - subright protocols



Communication channel [DOI] does not need to be integrity protected. Indeed, all transmitted values are also known by I , and I may, according to our assumptions presented in Section 2.2, freely exchange this information.

IssueRevTags. This protocol is used by a delegator to retrieve n additional revocation tags for the sub-rights of a certain main-right. It can be executed multiple times and proceeds as follows. First, Do creates a commitment C_{rev} containing random values e_{1u}, \dots, e_{nu} . This commitment, together with the main-right's revocation tag e and a verifiable encryption $encr$ of (e_{1u}, \dots, e_{nu}) are sent to RM . Do also proves that $encr$ is constructed correctly. After receiving random values e_{1i}, \dots, e_{ni} from RM , De 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 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, \dots, e_n which are constructed as $e_k = e_{ki} + e_{ku}$ for $k = 1, \dots, n$. Note that the resulting e_k 's are unknown to RM , and that their final value cannot be manipulated by Do .

Value n is fixed beforehand and determines the amount of retrieved revocation tags. In order to keep $Cred_{rev}$'s size limited, n should not be too large. Also, for similar reasons as in the *IssueRight* protocol, communication channel [DoRM] does not need to be confidentiality-protected.

DelegateRight. During the *DelegateRight* protocol, Do issues a t -show sub-right to De . The protocol may be preceded by an optional identification step from De to Do and is performed over a confidentiality-protected communication channel.

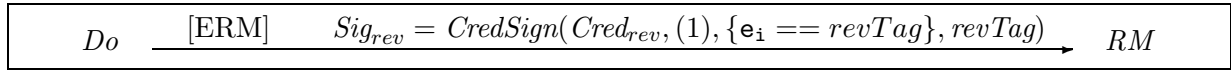
De creates a commitment C_{de} on her user secret s_{de} . She sends it to Do and proves that it is constructed correctly. Upon success, Do creates two commitments C_{do} and C_{sub} . C_{do} encodes her main-right's revocation tag e , while C_{sub} contains both the revocation tag e_i and the right-specifications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

- The signer owns credentials $Cred_{right}$ and $Cred_{rev}$.
- The same revocation tag e is encoded in both $Cred_{right}$ and $Cred_{rev}$.
- The rights encoded into C_{sub} are a subset of the rights encoded into $Cred_{right}$. (Here, we define a set of rights A to be a subset of another set B if its rights and validity periods are more strict than or equal to those specified in B .)
- C_{sub} 's attribute `subRevTag` is one of the revocation tags encoded in $Cred_{rev}$.
- t show-instances of $Cred_{right}$ are spent during the creation of Sig_{sub1} .

ShowRight. Due to their difference in structure, main-rights and sub-rights cannot be shown in a uniform way. Hence, the *ShowRight* protocol is split into separate protocols for both type of rights.

A base-right is shown by showing its corresponding credential $Cred_{right}$ and by proving that its revocation tag `revTag` does not belong to a blacklist BL . Additionally, Do proves that the rights required to access the service ($Nset$) are a subset of her encoded set of rights $Cred_{right}$.rights.

Figure 5: Revocation request for anonymous user's



A sub-right is a set of signatures (Sig_{sub1}, Sig_{sub2}) on a tuple $(C_{do}, C_{de}, C_{sub})$. During the show protocol, De provides V with a signature Sig_{show} which proves that De is the owner of user secret s_{de} encoded into C_{de} . Sig_{show} contains signatures (Sig_{sub1}, Sig_{sub2}) . In order to prevent replay attacks, Sig_{show} also contains some verifier-provided state information si . De then proves that her sub-right has not been revoked and that it contains sufficient rights for accessing V 's services. V checks the signatures and publishes Sig_{show} as a proof that one additional instance of the right is spent. She also checks if the sub-right is spent more times than allowed. Note that this last check can be deferred until after the protocol. In this case, deanonymization options must be added, such that De 's identity can be revealed should overspending be detected.

RevokeRight. A right is revoked by adding its revocation tag to a public blacklist BL . If $revTag$ belongs to a main-right, all rights in its revocation tree are revoked by retrieving the signatures Sig_{rev} on $revTag$ and by decrypting the as such found encryptions $encr$. 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.

As explained in Section 4.1.2, revocation requests by unidentified entities must be backed up with a proof that this entity is also the issuer of the right. For anonymous users, this can be accommodated by the protocol in Figure 5.

4.2.2 Evaluation

Functionality requirements.

- F1. Ad-hoc delegation is trivially achieved.
- F2. Constraints on the usage of a right ($F2a$), such as validity periods and limitations of the right, can naturally be encoded into the `rights` attributes. V checks the adherence to these constraints during the *ShowRight* protocol. Next to this, a limit can be set on the maximal number of usages of a right. For sub-rights, the fact that this limit is not exceeded can be checked by V either on-line during the *ShowRight* protocol or off-line when the *ShowRight* protocol is already finished. In the latter case, deanonymization options must be included to ensure the identification of the right's owner in case she overspent the sub-right.

The current construction of Figures 3 and 4 does not support any additional constraints on the delegation process ($F2b$). Nevertheless, constraints on Do 's delegation capabilities can be encoded.

- The maximal number of delegations performed by Do can be fixed. For this, an additional show-mode (here named the delegation-mode) is added to credential $Cred_{right}$. The maximal number of show instances in this new mode equals the maximal number

of delegations allowed for Do . Whenever a sub-right is issued, $Cred_{right}$ is shown t times in original mode and one time in delegation-mode. During the $ShowRight$ protocol, Do shows $Cred_{right}$ only in the original mode.

- Since the system itself is limited to rights issued by I and to direct sub-rights thereof, no maximal depth for the delegation tree must be encoded into the credential. Main-rights will have a delegation tree of depth at most one, while sub-rights will have no delegation tree at all.
- Currently, the only constraint on a sub-right is its requirement to be a subset of the rights encoded into $Cred_{right}$. More specific constraints can be encoded by including an additional attribute $SubrightsSpec$ in $Cred_{right}$. This attribute contains the most general specifications of an allowed sub-right, including maximal bounds on validity-periods. Whenever the $DelegateRight$ protocol is executed, instead of proving the relation $(rights \supseteq C_{sub}\text{-rights})$, Do then demonstrates the relation $(SubrightsSpec \supseteq C_{sub}\text{-rights})$.
- Constraints on the choice of sub-right owners can be encoded using similar techniques as described in Section 4.1.

Anonymity requirements.

- A1. Service access is anonymous, even if multiple entities collaborate and freely exchange their information. This applies both to main-rights and to sub-right. Unlinkability of service access is also achieved for main-rights. Sub-rights are linkable due to the uniqueness of tuple (Sig_{sub1}, Sig_{sub2}) . In order to overcome this problem, a t -show sub-right can be issued as t separate one-show sub-rights. Note, however, that this solution is rather expensive and that it fails for unlimited-show sub-rights. Finally, as mentioned before, right indistinguishability is not 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, some 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.

- A3. Delayed monitoring of main-rights can again be achieved using the pseudorandom function of Dodis and Yampolskiy [11]. It requires the following additions to the protocol.
 - During the $IssueRight$ protocol, Do picks a random value w and computes a verifiable encryption V_w of w as well as a commitment C_{do} containing w as attribute. She then sends V_w and C_{do} to I and proves the relation $V_w = VE(C_{do}, w)$ using the $CommProps$ primitive. If I accepts, an additional attribute $w := C_{do}.w$ is added to $Cred_{right}$.

- During the *DelegateRight* protocol, *Do* computes $y = F(w, i)$ for a value $i \in D$ (with D a predefined domain) which has not been used before. The extended tuple $(C_{do}, C_{de}, C_{sub}, y)$ is signed by *Do*. Additionally, during the creation of Sig_{sub1} , *Do* proves that y has been formed correctly.
- During the *ShowRight* protocol for the main-right, *Do* computes $y = F(w, i)$ for a yet unused value $i \in D$. *Do* sends y to *V* and proves to her that it is constructed correctly. If the *ShowRight* protocol is successful, *V* publishes y .

The *ShowRight* protocol for a sub-right is the same as in Figure 4. Note that, in this case, y is published as a side-effect of the publishing of Sig_{show} .

Monitoring of a sub-right is achieved by checking the verifier's logs for a tuple (Sig_{sub1}, Sig_{sub2}) . This monitoring is immediate, cannot be switched off and can only be performed by *Do*. Indeed, even though *V* (and hence any other entity) can link different usages of the same sub-right, she cannot link this sub-right to a particular issue protocol or to different sub-rights issued during the same interaction.

Security requirements.

- S1. Main-rights are unforgeable (*S1*) due to the unforgeability of credentials. Sub-rights are unforgeable due to the unforgeability of the *CredSign* signature scheme.
- S2. Only correct sub-rights are accepted by the verifier. During the *DelegateRight* protocol, *Do* explicitly proves that the sub-rights validity periods and right specifications are more strict than or equal to what is specified in the main-right. By loosing t show-instances of her main-right, she additionally proves her eligibility to issue a t -show sub-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 is discouraged by the use of non-transferable user secrets s_u . An exception to this is credential $Cred_{rev}$, which does not contain s_u . For this construct, 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. Multiple rights can be shown to the same verifier. These rights are either main-rights, sub-rights or a combination of both. Consistency of the rights can be demonstrated by an additional proof that the **Owner** attributes of the main-rights and the attribute **s** of commitments C_{de} in the sub-rights, are all the same value.
- S5. All rights are revocable and the revocation of a main-right implies the revocation of its corresponding sub-rights. In addition, unidentified entities can be prohibited from successfully requesting revocations.
- S6. Conditional deanonymization can easily be added uses the techniques described in Section 3.

Table 4: Communication channels for the signed warrant-based implementation

channel	authentication	integr.	confid.	anon.
[DoDe]	optional: De to Do	yes	yes	yes
[DeI]	I to De	yes	no	yes
[UV]	V to U	yes	no	yes
[ERM]	RM to E	yes	no	yes
	optional: E to RM			

Performance requirements. All protocols are relatively cheap and no additional entity is needed for the delegation. In contrast to the system of Section 4.1, the number of potential delegates does not affect the system’s performance in a meaningful way. It does, however, have its impact on the number of revocation tags needed by Do . In general, a good trade-off should be made between the number of tags in $Cred_{rev}$ and the expected number of delegates. The number of tags in $Cred_{rev}$ should be small enough to ensure the efficient execution of the *DelegateRight* protocol. On the other hand, it should be large enough to limit the number of executions of the *IssueRevTags* protocol.

4.3 New credentials

The following system is almost identical to the system presented in Section 4.2. This time, the signatures representing the sub-right can be exchanged with I for a new credential. Hence, all rights are represented using the same credential structure. Interesting properties resulting from this addition are delegation trees of arbitrary depth, unlinkable service access and right indistinguishability. Note that, due to the adoption of arbitrary depths of delegation trees, a main-right is not necessarily issued during an *IssueRight* protocol.

4.3.1 System

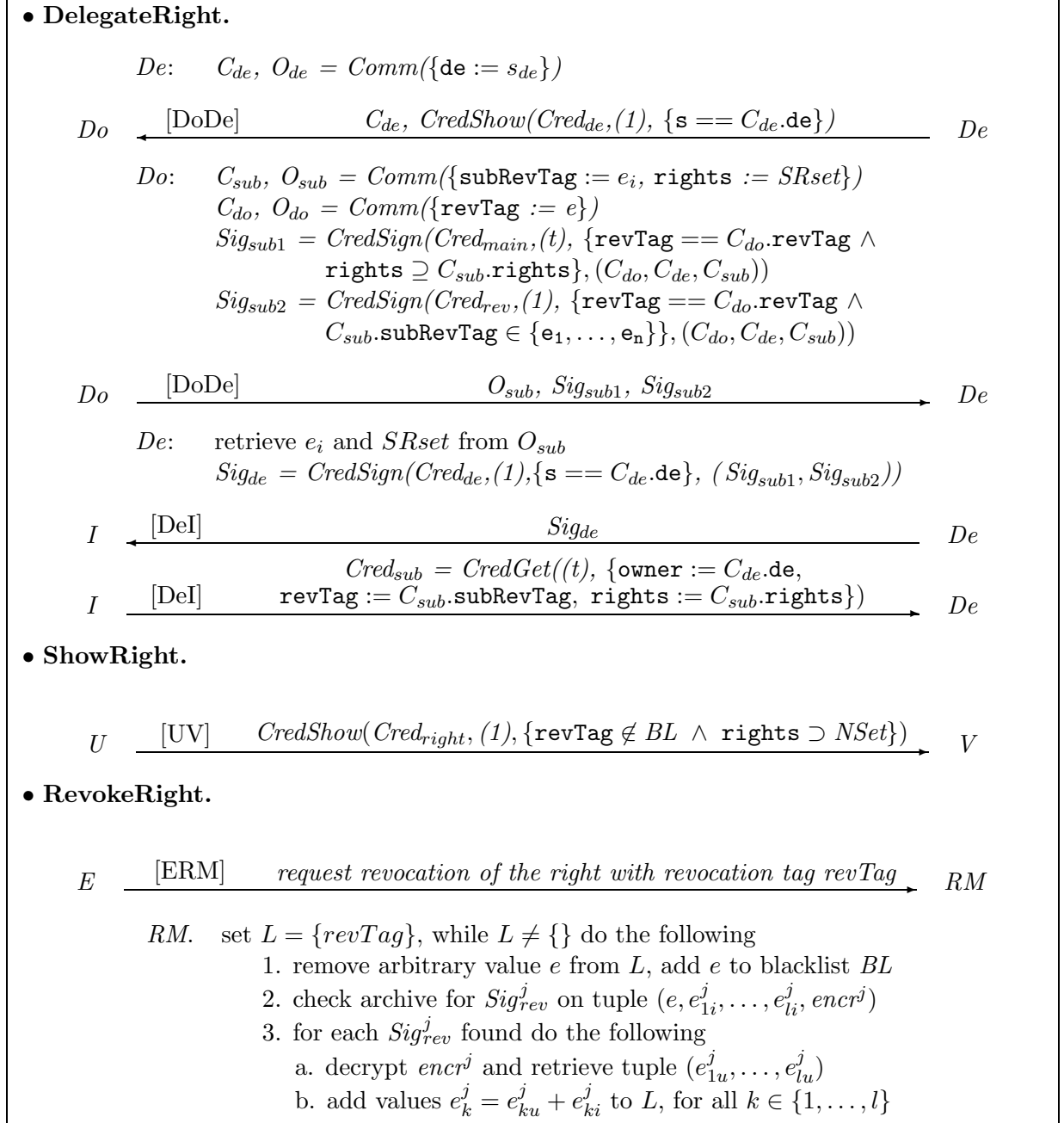
The system is depicted in Figure 6. The communication channels and their characteristics are summarized in Table 4. Protocols *Registration*, *IssueRight* and *IssueRevTags* are the same as in Section 4.2 and are, hence, not depicted in Figure 6. A credential representing a right is denoted as $Cred_{right}$, $Cred_{main}$ or $Cred_{sub}$, depending on its function as a general right (i.e. irrespective of its classification as a sub-right or a main-right), a main-right or a sub-right.

DelegateRight. The *DelegateRight* protocol consists of two phases which can be separated in time. It may be preceded by an optional identification step.

The first phase is performed over a confidentiality-protected communication channel and results in De having a signature tuple (Sig_{sub1}, Sig_{sub2}) on commitments $(C_{do}, C_{de}, C_{sub})$. The execution of this phase is identical to that of the *DelegateRight* protocol in Section 4.2.

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 user secret s_{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} .

Figure 6: New credential-based implementation of the delegation model



ShowRight. During the *ShowRight* protocol, U shows her credential $Cred_{right}$ to V . Additionally, she proves that it has not been revoked and that it contains sufficient rights for accessing V 's services.

RevokeRight. This protocol is based on the *RevokeRight* protocol in Section 4.2. As a right's delegation tree can have an arbitrary depth, the original protocol is extended to be iterative and to be repeated until all of the rights in the delegation tree are revoked.

4.3.2 Evaluation

Functionality requirements.

- F1. Ad-hoc delegation is trivially achieved.
- F2. Constraints on the usage of a right are identical as in Section 4.2. Constraints on delegation are also as in Section 4.2, except that now, a delegation tree can have an arbitrary depth. Issuer I can limit this depth by encoding an additional attribute **MaxDepth** into each credential $Cred_{right}$. Whenever the *DelegateRight* protocol is executed, Do determines a new value for this attribute and encodes it into C_{sub} . She also extends Sig_{sub1} with a proof of the relation $(\text{MaxDepth} > 0 \wedge \text{MaxDepth} > C_{sub}.\text{MaxDepth})$. Finally, during phase 2 of the protocol, I encodes $C_{sub}.\text{MaxDepth}$ into the resulting credential.

Anonymity requirements.

- A1. Service accesses are anonymous and unlinkable. Right indistinguishability is also achieved.
- A2. Properties and suggestion concerning sub-right unlinkability are the same as in Section 4.2.
- A3. Delayed monitoring of rights can be added using the verifiable pseudorandom function of Dodis and Yampolskiy [11]. A detailed description of the necessary protocol-additions is given below.
 - (a) During the *IssueRight* protocol, Do encodes a random value w into a new commitment C_{do} and provides I with a verifiable encryption V_w of w . She also proves that V_w is correctly formed based on $C_{do}.\mathbf{w}$. If this is successful, w is added as an additional attribute to $Cred_{right}$.
 - (b) During phase 1 of the *DelegateRight* protocol, De picks a random value v and encodes it as an additional attribute into C_{de} . She also computes a verifiable encryption $V_v = VE(v)$ and sends V_v to Do .
 Do then creates a value $y = F(w, i)$ for a yet unused value $i \in D$ and then signs the tuple $(C_{do}, C_{de}, C_{sub}, y, V_v)$ as before using the *CredSign*. In addition, she extends Sig_{sub1} with a proof that y is correctly formed.
 - (c) During phase 2 of the *DelegateRight* protocol, De proves to I that V_v is correctly formed. If successful, I adds value v as an additional attribute to $Cred_{sub}$. Note that, in order to perform this step, I does not need to know v 's value. Finally, I publishes De 's request Sig_{de} .

- (d) During the *ShowRight* protocol. U provides V with a value $y = F(x, i)$, for x the secret value (i.e. w for a right issued by I or v for a sub-right) encoded into U 's credential $Cred_{right}$ and for $i \in D$ with D a predefined domain. She also proves to V that this value is correctly formed. Note that a different value i must be used during each interaction.
- (e) Upon fulfillment of the tracking condition, I (or Do) iteratively requests the decryptions x of the relevant verifiable encryptions. She can then compute values $y = F(x, i)$ for all $i \in D$, and subsequently track all actions performed by the right or any of its sub-rights.

As value V_v encoded into De 's request Sig_{de} is publicly known, anyone can initiate the monitoring of this sub-right. In order to prohibit this unauthorized monitoring, a decryption request must be accompanied with a proof of (indirect) issuance of the right that is to be monitored. As an example for a sub-right determined by values $(C_{do}, C_{de}, C_{sub}, y, V_v)$, this can be done by proving knowledge of values i and w such that $F(w, i) = y$. Indeed, should an unauthorized entity be able to find values i and w such that $F(w, i) = y$, then she would also be able to break the pseudorandomness of $F(\cdot, \cdot)$.

Security requirements. Security requirements $S1, S2, S3, S5$ and $S6$ are satisfied for similar reasons as in Section 4.2. Consistency of rights ($S4$) is achieved by proving equality of the **Owner** attributes encoded in the credentials representing the different rights.

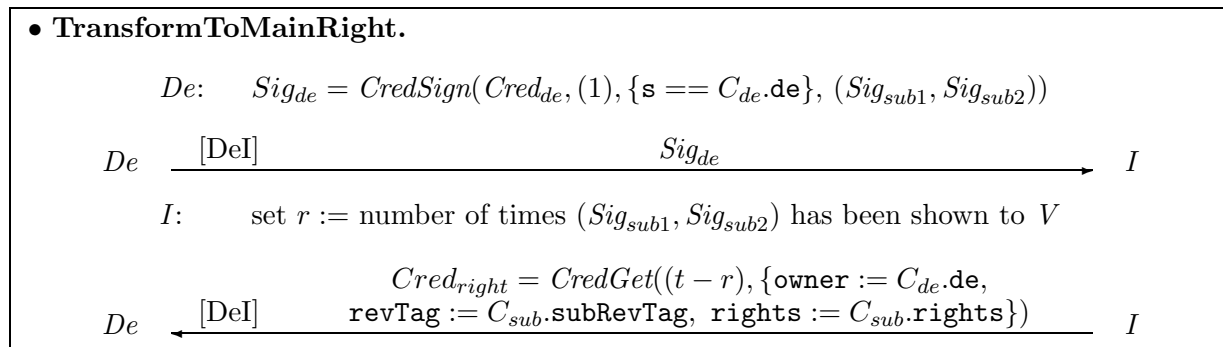
Performance requirements. The system's complexity is comparable to that of Section 4.2. One difference is the use of an additional entity I during the *DelegateRight* protocol. This results in a less efficient protocol, as De needs to perform an additional interaction with I in order to retrieve her sub-right. In the next section, we show how this problem can be alleviated in systems with more flexible anonymity requirements.

4.4 Hybrid System

Finally, we briefly discuss a hybrid system. The system combines the ideas of Sections 4.2 and 4.3, and as such enables a more flexible trade-off between performance and anonymity requirements. In particular, the same performance and anonymity characteristics are achieved as in the signed warrant system of Section 4.2. Next to this, additional anonymity can be acquired at the cost of one more protocol execution with I .

The system's structure and protocols are the same as in Section 4.2. One difference is the adoption of an additional protocol *TransformToMainRight*. This protocol enables U to transform sub-rights (Sig_{sub1}, Sig_{sub2}) into valid main-right credentials $Cred_{right}$. Its structure is depicted in Figure 7 and is based on phase two of the *DelegateRight* protocol of Section 4.3. The protocol is executed over an integrity protected and anonymous communication channel, and preceded by the authentication of I to De . First, De sends a signature Sig_{de} to I . This signature includes tuple (Sig_{sub1}, Sig_{sub2}) and additionally proves that De owns user secret s_{de} encoded into C_{de} . If this succeeds, and if (Sig_{sub1}, Sig_{sub2}) has not been shown to I before, I finds out how many times the sub-right has been shown to a verifier. For this, she counts the number of published signatures Sig_{show} containing (Sig_{sub1}, Sig_{sub2}) . Based on this number r and on the original show-limit t of the sub-right, I issues a new $(t - r)$ -show credential $Cred_{right}$. Additionally, she publishes

Figure 7: *TransformToMainRight* protocol



(Sig_{sub1}, Sig_{sub2}) as to ensure that it will not be accepted during future executions of the *ShowRight* protocol.

5 Comparison

A comparison of our techniques is given in Table 5. Note that the hybrid system of Section 4.4 is not included, as its properties depend heavily on the application for which it is used. In general, these properties are an application-specific combination of the features for a signed warrant system and a new credential system.

As can be seen from Table 5, each of the techniques has its own advantages and disadvantages. Hence, the choice of a system depends very much on the requirements of the application that is to be implemented.

The system based on transferable credentials is the least flexible of the four. It is also the only technique which does not enable sub-right unlinkability. On the other hand, it can be used very efficiently when the number of delegates is limited and when their identities are known beforehand. As an example for such applications, we think of a loyalty card or a membership card which can be used by all members of the same family.

The system based on signed warrants is very flexible and additionally enables sub-right unlinkability. The disadvantages of this technique are the poor linkability properties of sub-rights with respect to verifiers and to main-right owner Do . Therefore, its best usage is in applications where sub-rights can be shown only a limited number of times and where tracking by Do does not impose any major problems. An example of such an application might be a drug prescription. As the patient is too ill herself, she will assign another user to pick up the medicines at the pharmacy.

The system based on new credentials is very general. It meets all flexibility and privacy requirements of Section 2.3. Complexity goes up a little, however, as an additional entity is needed for the issuing of sub-rights. This system can best be used for applications with very broad requirements. An example of this can be the complex access structure of a medical database. Medical doctors retrieve access to the general database. Additionally, they can grant more limited accesses to specific parts of the database to their assistants or nurses.

Finally, the hybrid system combines all characteristics of signed warrants and of the new cre-

dential technique. It can be used in all settings for which the new credential technique can be used. In addition, the user has more flexibility when it comes to trading in anonymity for performance.

Table 5: credential-based implementations of the delegation model - comparison

	transferable creds	signed warrants	sub-right creds
ad-hoc delegation	no	yes	yes
expressiveness of constraints (1) on usage (2) on delegation	(1) validity prds/rights (2) only base and sub-rights, limits on nb/type of subr. and on choice of subr. owner	(1) validity prds/rights (2) only base and sub-rights, limits on nb/type of subr. and on choice of subr. owner	(1) validity prds/rights (2) limits on nb/type of subr, on choice of subr. owner, and on length of deleg. chain
privacy-preserving show protocol	anon. and unlinkable service access, rights indistinguishability	anon. service access, unlinkability only for base rights, no rights indistinguishability	anon. and unlinkable service access, rights indistinguishability
sub-right unlinkability	no	unlinkability until revocation time	unlinkability until revocation time
monitoring of rights	optional delayed monitoring of rights	optional delayed monitoring of base-rights, mandatory immediate monitoring of sub-rights	optional delayed monitoring of right
security	all security requirements (S1-S5) are satisfied	all security requirements (S1-S5) are satisfied	all security requirements (S1-S5) are satisfied
complexity	raises with nb of delegates, no additional entity needed	good, no additional entity needed	good, additional entity needed

References

- [1] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures (extended abstract). In *EUROCRYPT*, pages 591–606, 1998.
- [2] Michael Backes, Jan Camenisch, and Dieter Sommer. Anonymous yet accountable access control. In *WPES*, pages 40–46, 2005.
- [3] S. A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000. Available for free download from http://www.credentica.com/the_mit_pressbook.php.
- [4] Emmanuel Bresson and Jacques Stern. Proofs of knowledge for non-monotone discrete-log formulae and applications. In *ISC*, pages 272–288, 2002.

- [5] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, pages 93–118, 2001.
- [6] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In *ASIACRYPT*, pages 331–345, 2000.
- [7] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT*, pages 302–321, 2005.
- [8] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.
- [9] Jan Camenisch, Dieter Sommer, and Roger Zimmermann. a general certification framework with applications to privacy-enhancing certificate infrastructures.
- [10] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT*, pages 125–142, 2002.
- [11] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography*, pages 416–431, 2005.
- [12] Oded Goldreich. *Foundations of Cryptography*, volume Basic Tools. Cambridge University Press, 2001.
- [13] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.