Using Authority Certificates to Create Management Structures *

Babak Sadighi Firozabadi[1], Marek Sergot[2], and Olav Bandmann[1]

[1] Swedish Institute of Computer Science (SICS)
{babak,olav}@sics.se
[2] Imperial College of Science, Technology and Medicine
mjs@doc.ic.ac.uk

Abstract. We address the issue of updating privileges in a dynamic environment by introducing *authority cerrtificates* in a Privilege Management Infrastructure. These certificates can be used to create access-level permissions but also to *delegate* authority to other agents, thereby providing a mechanism for creating management structures and for changing these structures over time. We present a semantic framework for privileges and certificates and an associated calculus, encoded as a logic program, for reasoning about them. The framework distinguishes between the time a certificate is issued or revoked and the time for which the associated privilege is created. This enables certificates to have prospective and retrospective effects, and allows us to reason about privileges and their consequences in the past, present, and future. The calculus provides a verification procedure for determining, given a set of declaration and revocation certificates, whether a certain privilege holds.

1 Introduction

Many applications require a decentralised management of access permissions to their resources. We have identified the following kinds of applications in which management of access permissions should be decentralised.

- Applications operating in a highly dynamic environment, such as adaptive networks, where access permission updates have to be done frequently, locally, and partly automatically.
- Applications in which the administration of access permissions becomes so heavy that it affects the core business activities.
- Cases where security administrators are not fully trustworthy, and are potential security threats, whether deliberately or unintentionally.

To address the issue of updating access permissions, each organisation may define its own management structure. A management structure is normally a hierarchical structure defining how authorities and responsibilities are, or can

* This research is funded by Microsoft Research, Cambridge, UK.

be, distributed within an organisation. In [MS91], the authors identify four typical roles for a management structure, namely User, Security Administrator, Manager, and Owner. An authority can be delegated within a domain using a predefined management structure. The idea is that the owner of an object has full authority concerning access and disposition of his object, and he can also delegate these authorities to managers. A manager defines a set of users and a set of objects as the administrative scope of a security administrator. The security administrator has the authority to give permissions to the users of his predefined domain to access the objects in this domain. Notice that a security administrator may or may not be part of the scope of his administration, which means that he may or may not be able to give himself access permissions.

In the current paper, we generalise the idea of management structure, because in real world scenarios there is a need for creating different types of management structures, and because the management structures may themselves be subject to frequent changes.

In [FS99] we distinguish between having a permission and being institutionally empowered, within a given organisation or management structure, to create a permission. In this paper, we employ the term 'authority' in place of 'institutional power' (because the term 'power', which has a technical meaning in this context, can also have unintended connotations). We use this notion of 'authority' as a prerequisite for creating and changing management structures as well as for creating and deleting permissions. We use the term 'privilege' as a general term to cover both 'authority' and 'permission'.

Separating the concept of authority from the concept of permission allows us to represent scenarios in which an agent has the authority to create a privilege (a permission or an authority) without having that privilege himself, or without having the authority to create that privilege for himself.

2 Delegation

In the information security literature, *delegation* normally describes the act of distributing access permissions to agents in a distributed system. Here, we allow for delegation of *privileges*, that is, for delegation of authorities as well as permissions. We distinguish between two possible kinds of delegation:

1. **Delegation as creation of new privileges:** The delegatee receives his own privilege which is independent of the delegator's privilege in the sense that if the delegator's privilege is revoked, then it does not necessarily mean that the delegatee's privilege is revoked. In this case the delegation is the act of issuing a new privilege. An agent may be an authority to create a privilege for another agent without having that particular privilege himself, or even without being an authority to create that privilege for himself. *Transfer* of a privilege can be seen as a creation of a new privilege and revocation of an old one.
2. **Delegation by proxy:** The delegatee does not receive his own privilege, but he can exercise the privilege of the delegator, in the sense that he *speaks*

for or *acts on behalf of* the delegator. In this case, if the delegator's privilege is revoked then the delegatee cannot exercise that privilege any more.

Some applications may require support for both kinds of delegation, and a framework capturing both would provide a flexible treatment for management of permissions. However, in this paper we will focus only on delegation of the first type.

3 Attribute Certificates and Privilege Management Infrastructure

Attribute Certificates (AC), sometimes called Privilege Attribute Certificates (PAC), have been proposed in various forums for securely providing privilege information using public key technology. The main proposal for use of AC is for distribution of authorisations [FPD99]. Beside this, ACs can also be used for other purposes such as group and role assignment as suggested in [HBM98], and for qualification certificates as suggested in [WP]. Similar to the need for public key infrastructure (PKI) for use of public key certificates (PKC), there is a need for an infrastructure for the use of attribute certificates. In [FPD99] this infrastructure is called the Privilege Management Infrastructure (PMI).

There are several reasons for decoupling an attribute certificate from a public key/identity certificate. For example:

- An agent's attributes (privileges) change more often than the public key associated to his identity.
- The authority issuing attribute certificates is usually not the same as the authority issuing public key certificates.

In PKI models, the public key certificate authorities (CAs) are usually large institutions at national or even international level, which are trusted or legally empowered to issue identity certificates. The structure formed by the relations between the CAs is fairly static and globally recognised by users of the PKI system.

Here, we argue that in contrast to the PKI model, in the PMI model the management structures for attribute authorities (AAs) can be highly dynamic and mainly determined locally, i.e. at organisational level. Being an authority to create a privilege is itself a privilege that is subject to change.

We propose the use of attribute certificates for delegating privileges and creating management structures. A certificate is a signed and time-stamped statement, which can be seen as an illocutionary act with a declarative force performed by its issuer[1].

[1] In speech act theory (see e.g. [Sea69]), one distinguishes between different types of illocutionary acts. Here, we are mainly concerned with one type of illocutionary act, viz. declarative acts, or illocutionary acts with declarative force. The performer of a declaration (an illocutionary act with a declarative force) brings about the

The issuing of a certificate can be seen as a declaration made by its issuer to bring about the propositional content given in the certificate. Notice that this type of certificate will not be effective unless its issuer has the authority for its content. A certificate issued by an agent without the necessary authority can be seen as an unsuccessful attempt by its issuer to declare its content.

The main components of an attribute certificate are:

- Issuer (the distinguished name, or the public key of the issuer)
- Subject (the distinguished name, or a pointer to the subject's public key certificate, i.e. its serial number)
- Attribute (the set of attributes that are associated to the subject)
- Validity Interval (the time-interval within which the given attributes are said to be valid)
- Signature (the digital signature algorithm used to sign the certificate)
- Certificate Serial Number (a unique ID number for the certificate, assigned to the certificate by its issuer)

An attribute certificate of this type says that the issuer is declaring that the subject has the set of attributes listed in the attribute field. The content of certificates can be a proposition stating various things, e.g. that the subject belongs to the group of administrators, that the subject is assigned the role of senior manager, that the subject is 20 years old, that the balance of the subject's bank account is £100, and so on. The validity interval field indicates the period of time for which the attributes hold for the subject.

We call an attribute certificate in which the issuer assigns some authority (or institutional power in the terminology of [FS99]) to the subject of the certificate as an *authority certificate*. An authority certificate can be used by an agent to delegate some authority to others, as for example when the owner of an object delegates, to some managers, the authority to create permissions to his object.

An authority certificate may be used to create an authority to initiate several chains of authority delegations. By expressing constraints on future delegations one defines the scope of future management structures in an organization [BDF01]. However, any delegation chain must originate from a *source of authority* for what is delegated. Who is recognized as a source of an authority, and in what conditions, is a policy issue and is application–domain specific. Different applications may have different policies for recognising sources of authorities. In many applications, but not all, the owner(s) of a resource are recognised as sources of authority for permissions and authorities concerning that resource, for example.

proposition that is the content of his declaration, if, and only if, he has the required authority for doing so. It is also possible to view some certificates as *assertions*, that is, in speech act terms, as illocutionary acts that assert the truth of a proposition without necessarily creating it. However, in this present framework nothing is gained from making the distinction and so we choose to treat all certificates as having declarative illocutionary force.

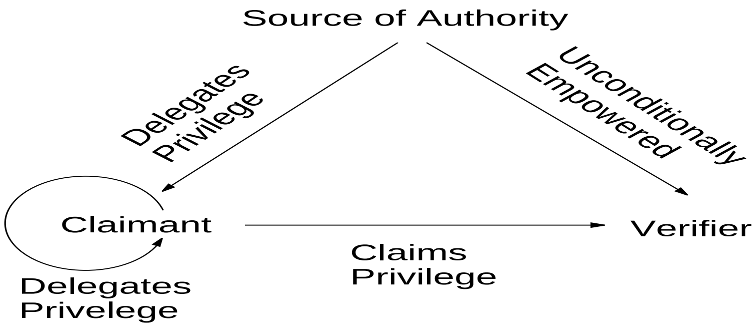

Fig. 1. The Control Model

Figure 1 shows the control model given in [FPD99] in which the claimant receives a privilege, directly or indirectly, from the source of authority. In order for the claimant to exercise his privilege (his access permission), the verifier needs sufficient credentials, in the form of certificates, to verify the claimant's privilege. Note that there may be a number of intermediary authorities between the source of authority and the final claimant of the privilege. This means that the set of certificates provided to the verifier may contain a number of authority certificates showing a proper delegation chain originating from the source of authority and leading to the claimant.

There are at least two possible models for the control model:

1. *Centrally updated model*: Any delegation step is reported directly to the verifier, by sending the authority certificates to the verifier, who updates the existing management structures in its database.
2. *Distributed model*: At the delegation step, the delegatee (the claimant) receives his new privilege and all the intermediate authority certificates originating from the source of authority to the delegator. The claimant provides this set of certificates to the verifier at the time of his privilege request.

Each of these models has a number of advantages and disadvantages making them suitable for different kinds of applications. We will not discuss the issues associated to each model in this paper. However, in both models the verifier needs a mechanism for deciding, given a set of certificates, whether the claimant's privilege holds. In the next section, we describe a calculus that can be used by the verifier for reasoning about privileges and delegated authorities.

4 The Framework

In this section we present a framework for a privilege management system using attribute certificates. The issuing of certificates is the only type of action considered in this framework. Issued certificates are submitted to a privilege verifier as shown in figure 1.

The privileges managed by the verifier are of the following two types.

- *Access-level permission* (e.g., permission to read or write a file, or permission to execute a program).
- *Management-level authority* (i.e., authority to declare an access-level permission, or authority to declare a management-level authority).

Here, we consider only two types of certificates, declaration and a simple form of revocation.

- Declaration certificates are represented as:

$$declares(issuer, p[I], \textit{time-stamp}, id).$$

We interpret a declaration certificate as an action description for a declaration performed by its *issuer* at time *time-stamp* to bring about that privilege p holds during time interval $[I]$. The *id* is the unique id of the certificate, either generated by its issuer or generated by the privilege management system which the verifier is a part of. Validation of signatures is of course an essential component of verifying a certificate, but signatures are not part of the reasoning process for verifying that a privilege holds, and for this reason signatures do not appear in the representation of certificates.

- Revocation certificates are represented as:

$$revokes(issuer, id, \textit{time-stamp}).$$

Note that a revocation certificate does not have an *id* itself, but it contains the *id* of the certificate that it is revoking. In the present framework, we do not allow revocation of a revocation certificate. Of course, one can imagine scenarios in which there is a need for recovery from earlier revocations. However, in the current framework we do not consider this type of scenario.

4.1 Semantics of the Calculus of Privileges

Informally, the idea is that a privilege p holds at a time-point t when there is a certificate C declaring that p holds for some interval I containing t; the certificate C moreover must be 'effective' at t, in the sense that it was issued by s at a time when s had the authority to declare p to hold for interval I. The authority of s, in turn, requires a certificate that was effective at the time C was issued — and so on, in a chain of effective certificates back to some source whose authority can be accepted without certification (as determined by the organisational structure).

The following definitions make these ideas precise. The complication is that we are here dealing with two levels of time — the time at which a certificate is issued, when it can be effective or not, and the time at which a given privilege holds or not. It it important to notice that we do *not* require that a certificate declaring privilege p for time interval I must be issued before I. In our scheme, a certificate can create a privilege *retrospectively*. We comment further on this and other features after presenting the definitions.

Definition 1. Let *AGN, ACT*, and *OBJ* be the sets of agents, actions, and objects, repectively. We define the set of privileges Φ as:

- $perm(s,a,o)[I] \in \Phi$, if $s \in AGN,\ a \in ACT, and\ o \in OBJ$;
- $pow(s,\phi)[I] \in \Phi$, if $s \in AGN, and\ \phi \in \Phi$.

We define the set of declaration certificates Σ^+ and the set of revocation certificates Σ^- as:

- $declares(s,\phi,t,id) \in \Sigma^+$, if $s \in AGN,\ \phi \in \Phi,\ t \in \mathbf{R},$ and $id \in \mathbf{N}$, where $\mathbf{R}$ denotes the real numbers, and $\mathbf{N}$ denotes the natural numbers;
- $revokes(s,id,t) \in \Sigma^-$, if $s \in AGN,\ id \in \mathbf{N},$ and $t \in \mathbf{R}$.

Privileges of the form $perm(s,a,o)[I]$ denote access-level permissions, while privileges of the form $pow(s,\phi)[I]$ denote management-level authorities.
(In the definitions above $[I] = [t_{start}, t_{end}]$, where $t_{start} \in \mathbf{R}$, $t_{end} \in \mathbf{R}$ and $t_{start} \leq\ t_{end}$.)

Definition 2. We define a certificate database to be a tuple $\mathcal{D} = (\mathbf{SoA}, \mathbf{D}^+, \mathbf{D}^-)$, where $\mathbf{SoA} \subset \Phi$ is a finite set of *Source of Authority* privileges, $\mathbf{D}^+ \subset \Sigma^+$ is a finite set of declaration certificates and $\mathbf{D}^- \subset \Sigma^-$ is a finite set of revocation certificates. We adopt the following constraints on a certificate database.

1. If $declares(s_1,\phi_1,t_1,id) \in \mathbf{D}^+$, and $declares(s_2,\phi_2,t_2,id) \in \mathbf{D}^+$, then $s_1 = s_2,\ \phi_1 = \phi_2$, and $t_1 = t_2$,

   This says that $\mathbf{D}^+$ cannot contain two different certificates with the same id.

2. If $declares(s_1,\phi,t_1,id) \in \mathbf{D}^+$ and $revokes(s_2,id,t_2) \in \mathbf{D}^-$, then $s_1 = s_2$ and $t_1 \leq t_2$.

   This says that a certificate can be revoked only by its issuer and not before it is declared. In fact, the first restriction can be relaxed but this introduces the need for extra components which are omitted here for simplicity.

3. If $revokes(s_1,id,t_1) \in \mathbf{D}^-$ and $revokes(s_2,id,t_2) \in \mathbf{D}^-$, then $s_1 = s_2$ and $t_1 = t_2$.

   This says that there cannot be two revocations of the same declaration certificate in the same database. We adopt this restriction to simplify the database in order to streamline the theory.

Definition 3. Let $\vdash$ be the *validates relation* between a privilege and a declaration certificate, where

$$pow(s,\phi)[I] \vdash declares(s,\phi,t,id),\ \text{if } t \in [I];$$

and, if $\Gamma \subseteq \Phi$, then

$$\Gamma \vdash d,\ \text{if } \exists\ q \in \Gamma \text{ such that } q \vdash d.$$

Definition 4. We define the set of *effective* declaration certificates $\mathbf{E}_{\mathcal{D}}(t) \subseteq \mathbf{D}^{+}$ of a database $\mathcal{D}$ at a certain time t, as:

$$\mathbf{E}_{\mathcal{D}}(t) = \{declares(s, p[I], t_1, id) \in \mathbf{D}^{+} \mid t \in [I] \ \& \\ (revokes(s, id, t_2) \in \mathbf{D}^{-} \rightarrow t_2 > t)\}.$$

Definition 5. Let $d_1, d_2 \in \mathbf{D}^{+}$, where $d_1 = declares(s_1, \phi_1, t_1, id_1)$ and $d_2 = declares(s_2, \phi_2, t_2, id_2)$. We define the *supports* relation $S_{\mathcal{D}}$ as follows:

$$d_1 \ S_{\mathcal{D}} \ d_2 \text{ if } d_1 \in \mathbf{E}_{\mathcal{D}}(t_2) \text{ and } \phi_1 \vdash d_2.$$

Definition 6. The set of certificate chains $C_{\mathcal{D}}$ in a certificate database $\mathcal{D}$ is the transitive closure of $S_{\mathcal{D}}$.

Note that, $C_{\mathcal{D}}$ at a time-point t may contain chains that are no longer of use; chains that can be extended with further certificates; and chains that are dormant (see figures in the following section).

Definition 7. We define the set of true privilege statements at a time-point t, in our calculus of privileges, by defining function $h_{\mathcal{D}} : \mathbf{R} \rightarrow 2^{\Phi}$ as:

$$\begin{aligned} h_{\mathcal{D}}(t) = \{p \mid \ & p[I] \in \Phi \ \wedge \\ & (p[I] \in \mathbf{SoA} \ \vee \\ & (d_1, declares(s, p[I], t_2, id)) \in \mathbf{C}_{\mathcal{D}} \wedge declares(s, p[I], t_2, id) \in \mathbf{E}_{\mathcal{D}}(t) \ \wedge \\ & \mathbf{SoA} \vdash d_1)\}. \end{aligned}$$

We also say that a privilege p *holds* at time-point t when $p \in h_{\mathcal{D}}(t)$.

4.2 Examples

In this section we present some diagrams to illustrate the formal definitions just given.

Each horizontal line of the diagram represents a (declaration) certificate. The vertical arrows depict the times at which certificates were issued. The shaded rectangles show the time intervals of the privileges declared by the certificates. For simplicity, the examples show only one privilege for each certificate, though this is not a restriction of the framework. Short vertical bars depict revocations; in Figure 2, the certificates issued at d_3, d_2, and d_4 have been revoked. The certificates issued at d_3 and d_2 were revoked before the associated privilege intervals expired, as indicated by the lighter shading. The certificate issued at d_4 declared a privilege for an interval which begins before the certificate was issued. The framework allows certificates to make retrospective declarations. Although not shown in this figure (but see Figure 3), it is possible that a certificate issued at time-point t could declare a privilege that holds for an interval entirely in the past of t.

The arrangement of the vertical arrows is intended to illustrate the *supports* relation between certificates. So the certificate issued at d_1 (which was issued

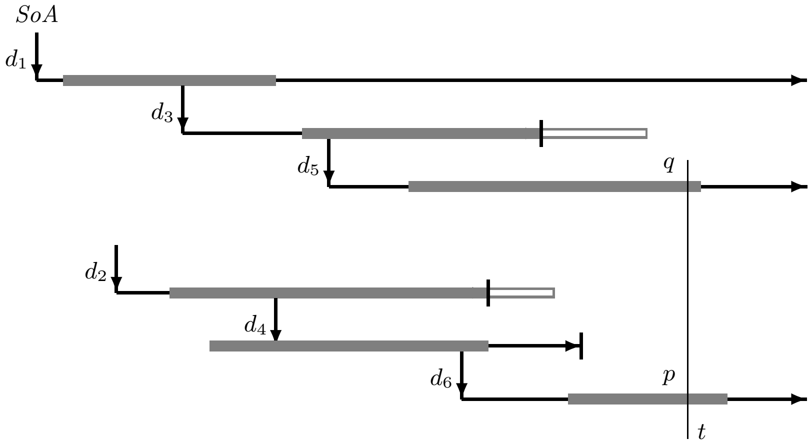

Fig. 2. Two certificate chains

by some source of authority SoA) supports the certificate issued at d_3, which in turn supports the certificate issued at d_5. The chain (d_1, d_3, d_5) is *rooted* since the certificate issued at d_1 is rooted (it was issued by a source of authority, we are supposing). The chain (d_2, d_4, d_6), on the other hand, is not *rooted*: d_2 is not issued by a source of authority, nor supported (we are supposing) by a rooted certificate. We call such chains *dormant chains.* Therefore, the privilege q declared by the certificate issued at d_5 holds at the time-point t shown in the diagram, but the privilege p declared by the certificate issued at d_6 does not hold at t (assuming there are no other chains besides those shown in the diagram).

Because certificates can have retrospective effects, a dormant chain can become rooted as a result of a later declaration. This is illustrated in Figure 3, where the previously dormant chain (d_2, d_4, d_6) becomes rooted as a result of the issuing of the declaration certificate at d_7.

Retrospective effects of this kind can be used to implement a type of *approval mechanism.* In the example, the issuer of certificate d_2 creates one or more chains of privilege-creating certificates. These remain dormant until eventually made effective ('approved') by the issuing of a suitable certificate at d_7.

Some observations:

- revoking a certificate declaring $p[I]$ before the interval I has started means that this certificate can never be used to create a chain (the privilege p can never be exercised on the basis of this certificate);
- revoking a certificate declaring $p[I]$ after the interval I has ended has no effect — any chain created using this certificate is not destroyed by the revocation;
- more generally, revocation of a certificate that has already been used to create a chain will not affect the chain — 'what's done is done', according to the specific notion of revocation supported in the present framework.

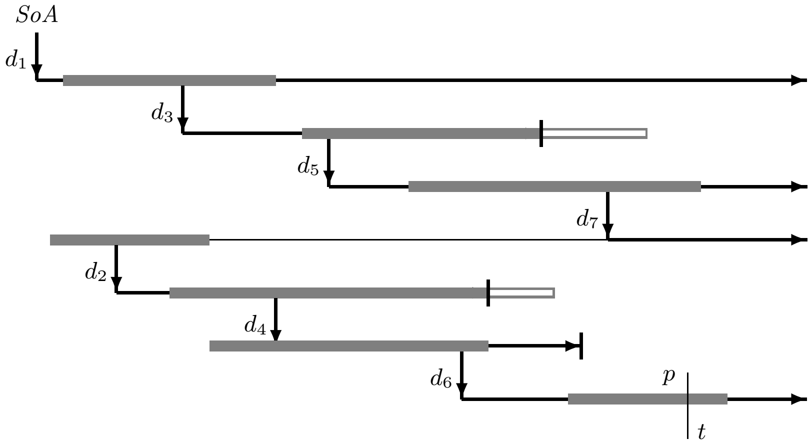

Fig. 3. A rooted certificate chain

As observed in the introductory section, it is also possible to conceive of other forms of revocation which can undo past effects. These forms of revocation are not discussed further in this paper, but will be presented in an extended version of the framework in future work.

4.3 The Calculus of Privileges

In this section we present a logic program which implements the semantics given above. The predicate $holds(P,T)$ is used to query the system to determine whether a privilege P holds at time-point T given a set of certificates database. (The program can be executed as a Prolog program as it stands, once the symbol ':' is declared as an infix functor).

[PC 1.] $holds(P,T) \leftarrow C = declares(S, P{:}[T_s, T_e], T_0, ID),$
$\quad effective(C,T),$
$\quad T_s \leq T \leq T_e.$

[PC 2.] $effective(C,T) \leftarrow C = declares(S, P{:}[T_s, T_e], T_0, ID),$
$\quad rooted(C),$
$\quad T_0 \leq T,$
$\quad not\,[revokes(S, ID, T_1),\ T_1 \leq T].$

[PC 3.] $rooted(C) \leftarrow chain(C1, C),$
$\quad C_1 = declares(S, P, T_0, ID),$
$\quad sourceOfAuthority(S, P).$

[PC 4.] $chain(C, C).$

[PC 5.] $chain(C1, C2) \leftarrow supports(C1, C2).$

[PC 6.] $chain(C1, C2) \leftarrow supports(C1, C3),$
$\qquad chain(C3, C2).$

[PC 7.] $validates(pow(S, P){:}[T_s, T_e], C) \leftarrow C = declares(S, P, T, ID),$
$\qquad T_s \leq T \leq T_e.$

[PC 8.] $supports(C1, C2) \leftarrow C1 = declares(S_1, Q{:}[T_s, T_e], T_1, ID_1),$
$\qquad C2 = declares(S_2, P, T_2, ID_2),$
$\qquad validates(Q{:}[T_s, T_e], C2),$
$\qquad not\,[revokes(S_1, ID_1, T_3),\ T_3 \leq T_2].$

In this program it is assumed that there is an up-to-date *source of authority* database, and that a source of authority privilege is created using a declaration certificate issued by the source of authority of that privilege.

The program can be generalised very easily to define a predicate $holds(P, T, T_D)$ representing that, according to the certificates issued up to and including time T_D, privilege P holds at time T. This generalized form allows one to query not only the current state of the set of certificates database, but all past states as well. The required modification is very straightforward. Details are omitted here.

5 Implementation issues

The scheme presented in the preceding sections supports many different models. For example, in a centralized system (ref. Figure 1), the *holds* relation can be materialized, that is, computed and stored for immediate look-up as required by the verifier, and updated incrementally whenever a new declaration or revocation certificate is received. There are well-established techniques for executing logic programs in this fashion. The database of all declaration and revocation certificates can also be queried to determine which privileges held at which times in the past, which may be useful for, e.g., auditing purposes.

In a distributed model, the 'claimant' presents a portfolio of certificates, which provide a set of certificates database on which the verifier can execute the reasoning calculus directly. Here there are several further options for the treatment of revocations. In one model the verification engine generates requests to a trusted revocation server as required. In another possible model, the verification engine checks locally against a list of revocation certificates broadcast from time to time by the revocation server. There are many other possible combinations. The point is that the same reasoning mechanism presented in the preceding section can be applied in each case.

Although we have presented the reasoning engine as a logic program, which can be executed in Prolog or in some other logic programming system, it is also easy to re-code the algorithm in another database formalism or programming language if that is preferred. We leave detailed discussion of such implementation techniques to another paper.

6 Conclusion and further extensions

We have addressed the issue of privilege management by using a type of attribute certificate that we call an authority certificate. We have made a distinction between two types of delegations — delegation as creation of new privileges, and delegation by proxy — though only the first of these is discussed in this paper.

We have presented a semantic framework and a calculus for reasoning about privileges based on a distinction between access level permissions and management level authorities. The calculus can be used by a verifier to check whether a certain privilege holds given a set of declarations and revocations. The framework supports flexible reasoning with time, such that certificates can be issued to create privileges in the past, present and future.

In the framework we present in this paper we have kept revocation certificates as simple as possible. Only the issuer of a declaration certificate can revoke it, and once revoked, a certificate cannot be reinstated. We are currently extending the framework by allowing more complex revocation certificates providing a richer set of revocation mechanisms. Finally, we intend to extend the framework with roles. These do not affect the core calculus but introduce a number of further choices which we are currently investigating.

Acknowledgement

We would like to thank Jason Crampton for suggesting a number of improvements to an earlier draft of this paper, which made the final draft tidier and easier to read.

References

[BDF01] Olav Bandmann, Mads Dam, and Babak Sadighi Firozabadi. Constrained Delegation. 2001. In preparation.

[FPD99] Final Proposed Draft Amendment on Certificate Extensions(v6). generated from Collaborative ITU and ISO/IEC meeting on the Directory, April 1999. Orlando, Florida, USA.

[FS99] Babak Sadighi Firozabadi and Marek Sergot. Power and Permission in Security Systems. In B. Christianson, B. Crispo, and M. Roe, editors, *Security Protocols*, number 1796 in Lecture Notes of Computer Science, pages 48–53, Cambridge, UK, April 1999. Springer Verlag.

[HBM98] R. J. Hayton, J.M. Bacon, and K. Moody. Access Control in an Open Distributed Enviroment. In *Proceeding of IEEE Symposium on Security and Privacy*, pages 3–14, Oakland, CA, 1998.

[MS91] J. Moffett and M. Sloman. Delegation of Authority. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management II*, pages 595–606. North Holland, April 1991.

[Sea69] John R. Searle. *Speech Acts*. Cambridge University Press, Cambridge, 1969.

[WP] Petra Wohlmacher and Peter Pharow. Applications in health care using public-key certificates and attribute certificates. In *Proceedings of the 16th Annual Computer Security Applications Conference 2000 (ACSAC 2000)*, pages 128–137, New Orleans, Dec. IEEE Press.