Revocation in the privilege calculus *

Babak Sadighi Firozabadi[1] and Marek Sergot[2]

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

Abstract. We have previously presented a framework for updating privileges and creating management structures by means of authority certificates. These are used both to create access-level permissions and to delegate authority to other agents. In this paper we extend the framework to support a richer set of revocation schemes. As in the original, we present an associated calculus of privileges, encoded as a logic program, for reasoning about certificates, revocations, and the privileges they create and destroy. The discussion of revocation schemes follows an existing classification in the literature based on three separate dimensions: resilience, propagation, and dominance. The first does not apply to this framework. The second is specified straightforwardly. The third can be encoded but raises a number of further questions for future investigation.

1 Introduction

A public key certificate (PKC) is a data record digitally signed by the private key of its issuer, a certificate authority (CA). A PKC can be seen as a statement by its issuer to certify that there is a binding between an identity (a distinguished name) and a certain public key. Revocation of a PKC can be seen as another statement saying that from a certain time point—the time-stamp of the revocation or the time-stamp of the revocation list containing the revocation—the binding given in the PKC no longer holds. The usual reason for revoking a PKC is that the private key matching the public key given in the certificate is lost or stolen.

Sometimes, in addition to the identity of the key holder, a PKC contains other information, e.g., the key holder's affiliation. As a consequence, a PKC sometimes has to be revoked not because the private key is lost, but because the affiliation of the key holder has changed. However, with the introduction of new types of digital certificates, i.e., *attribute certificates* (AC), there will often be cases in which one has to revoke a certificate because the attribute (privilege) given in the certificate needs to be deleted. For example, a certificate containing an access permission may get revoked because the permission given

* This research was supported in part by Policy Based Management Project funded by the Swedish Agency for Innovation Systems, and by AMANDA project funded by Microsoft Research in Cambridge, UK.

in that certificate does not hold any longer and not because the private key used to sign the certificate has been exposed.

In an earlier paper [FSB01] we presented a framework for updating privileges in a dynamic environment by means of *authority certificates* 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 presented 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.

In our earlier presentation we restricted attention to the simplest form of revocation only. The present paper expands the framework for managing authorities to support a richer set of revocation schemes. The ideas in this paper has earlier been presented in the position paper [FS02]. The current paper extends the ideas and incorporate them in the privilege calculus.

1.1 Revocation (deletion) Classification

In [HJPW01], the authors classify revocation schemes based on three dimensions—*resilience*, *propagation*, and *dominance*—which can be combined in various ways to provide several distinct categories. Although these authors do not consider revocation of *certificates*, but rather deletion of privileges that have been granted in a delegation chain, and although their way of representing privileges and delegations is not the same as ours, we nevertheless find it instructive to consider how that classification might apply to the classification of revocation schemes in our framework. The three dimensions are:

1. **Resilience**: This concerns how a privilege may be revoked in such a way that its effects are persistent, that is to say, in such a way that no other agent may subsequently re-create it. For example, this is the effect that is obtained by creating a 'negative privilege' that will always override its positive counterpart. In this way, the subsequent granting of the positive privilege will never have an effect, since it will always be overriden by the negative one.
2. **Propagation**: This concerns how revocation of a privilege may have indirect effects on other privileges stemming from it in a delegation chain.
3. **Dominance**: This concerns how an agent may be allowed to revoke privileges that are not directly delegated by him. For example, the case considered by [HJPW01] is one in which an agent retains the ability to revoke not only his own delegations, but those of any other agents whose privileges were obtained in a delegation chain stemming from him.

Of these three, the first, resilience, is not applicable to our framework. We do not support the granting of negative privileges. Put another way, our framework deliberately separates privileges from the means by which they are created (here, the issuing of certificates). It is meaningful in our framework to revoke *certificates*, and thereby indirectly delete privileges, but it is not meaningful to revoke privileges directly.

The other two dimensions, however, do apply, and are examined in the body of the paper.

2 Managing Authorities

In ITU-T Recommendation X.509 (2000) [X.500], the standard is extended to include Privilege Management Infrastructure (PMI) as well as Public Key Infrastructure (PKI). PMI is similar to PKI, but its purpose is to give an infrastructure for issuing and managing attribute certificates for assigning and conveying privileges.

The main components of a PMI are: Attribute Certificates (AC), Sources of Authority (SoA), Attribute Authorities (AA), and Attribute Certificate Revocation Lists (ACRL). An attribute certificate is, like a public key certificate, a digitally signed statement (in the form of a data structure) certifying the binding between the holder of the certificate and some of his privileges. A privilege in an attribute certificate can be, for example, an access permission, a group membership, or a role assignment. A SoA is an agent who is recognised by the users as initially empowered to create and delegate certain privileges. An AA is an agent who has been delegated, by the SoA, the authority to create a privilege. A number of AAs can create an *authority management structure* in which authorities have been delegated from the top AA, e.g. the SoA, to subordinates. An ACRL is a list of references of attribute certificates that are no longer to be considered valid.

The framework introduced in our earlier paper [FSB01] provides a means for creating and updating authority management structures such as an AA structure in the PMI model of X.509 (2000). In that paper we distinguish between an access level permission and a management level authority. $perm(s, a, o)$ represents an *access level permission* which says agent s is permitted to perform action a on object o. $auth(s, p)$ represents an *management level authority* which says agent s has the authority to bring about privilege p. Here, p can either be an access-level permission or another management-level authority. We use the term *privilege* to refer both to an access level permission and a management level authority. Note that having the authority to create a privilege does not necessarily mean that one has or can enjoy that privilege oneself; nor that one has the authority to create that privilege for oneself.

A certificate stating that its holder has the authority to create a privilege by means of issuing another certificate is called here an authority certificate. Neither in PMI [X.500] nor in other digital certificate frameworks there is a direct support for encoding management authorities in certificates, however one

can see an authority to create a privilege as an permission to delegate that privilege which can be encoded in attribute certificates.

In [FSB01], we considered only the simplest type of revocation, one that makes it possible to invalidate a certificate from the time the revocation occurs for all times in the future. In the current paper, we generalise this revocation type, allowing the revoker to disable a certificate in the past, present, or future, permanently or just for some specified period of time. This means that the time at which the revocation occurs and the time for which the revoked certificate becomes invalidated are independent. Furthermore, in the previous framework we allowed only the issuer of a certificate to revoke it (though it was possible to get other effects indirectly). In this paper we consider a number of alternatives, under the heading of 'dominance'.

2.1 The Framework of Authority Management

The framework contains only two type of actions: the issuing of certificates and the revoking of certificates.

– Certificates are represented as:

$$\mathit{certifies}(\mathit{issuer}, p[I], \mathit{time\text{-}stamp}, \mathit{id}).$$

The intended reading is as follows: the *issuer* makes an attempt, at time *time-stamp* to bring about that privilege p holds for the time interval $[I]$. We say that the certificate *certifies* (or sometimes 'contains') the privilege p, and we call $[I]$ the *validity interval* of the certificate. If p is a permission then the action in p is an access level action, e.g. read or write a file. If p is a management level authority, of the form $\mathit{auth}(s, q)$, then the action in p is the creation of a privilege q for s. The *id* is the unique identifier of the certificate.

For simplicity we leave out all the parts of a certificate management system that do not have any impact on the reasoning required to determine whether a given privilege holds at a given time. In particular, 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 our representation of certificates.

– Revocations are represented as:

$$\mathit{revokes}(\mathit{issuer}, \mathit{id}, [I], \mathit{time\text{-}stamp}).$$

Revocations, like certificates, are seen as time-stamped statements. In contrast to certificates, a revocation does not have an *id*, but it contains the *id* of the certificate which is the subject of the revocation. The interval $[I]$, called the *disabling interval*, represents the period of time for which the revocation disables the certificate with the id *id*. By specifying the disabling interval, revocation can be used to disable the particular instance of the privilege in that certificate either temporarily or permanently. Note that revocation works on *certificates*: if the

same privilege is created by several different certificates, revoking only one of them will not necessarily disable the privilege itself. All of the certificates would have to be revoked for that to happen.

Given a historical database of certificates and revocations, it is possible to determine whether a privilege p holds at a given time. Informally: 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).

Now, we give a number of definitions to make these ideas more precise. We assume that there is a historical database recording the issued certificates and their revocations. This database may be stored in a distributed form: the only requirement is that the reasoning engine for determining whether a certain privilege holds has access to the information.

Definition 1. *A certificate c_1 **directly supports** another certificate c_2 if*

1. *the privilege given in c_1 is the authority for the issuer of c_2 to bring about the privilege given in c_2 at the time of issuance of c_2, and*
2. *c_1 is not disabled at the issuance time of c_2.*

*If condition 1 holds we say that the privilege given in c_1 **validates** the certificate c_2.*

Note that this definition refers only to the time point at which c_2 is issued. If c_1 becomes disabled at any other time, that is to say, for some time period not containing the issuance time of c_2, the support of c_1 for c_2 will not be affected.

Definition 2. *A set of certificates $c_1 \dots c_n$ is a **chain** if each c_i directly supports c_{i+1}, for $1 \leq i < n$.*

A chain represents an authority management structure created by a number of authority certificates. A chain usually, but not always, ends in an end-entity attribute certificate containing an access level permission.

Definition 3. *A certificate c_i in a chain $c_1 \dots c_n$ **indirectly supports** a certificate c_j, if $i + 1 < j$ and $1 \leq i < n$.*

In any application there should be a way of defining who is a source of authority and in what circumstances. For example, in many applications the owner of an object is considered to be the source of authority for any privilege concerning that object. We assume that there is a specified way of recognising sources of authorities, either because the SoA relation between an agent and a privilege is given explicitly, or by means of a set of rules which defines this relation.

Definition 4. *A certificate is called* ***rooted*** *if it is issued by the source of authority of the particular privilege given in the certificate.*
A chain of certificates is ***rooted*** *if the first certificate in the chain is rooted.*

In this framework, anyone can issue a certificate at any time with or without having the necessary authority to make it effective. Without the necessary authority, the certificate has no effect. However, it is possible for a certificate c_1 to become supported, retrospectively, by another certificate c_2 issued at a time later than c_1. This happens when the validity interval of c_2 contains the issuance time of c_1. Examples of the use of such retrospective mechanisms are provided in [FSB01].

Definition 5. *A chain that is not rooted is called a* ***dormant chain****.*
A certificate is called ***dormant*** *at time t if it is part of a dormant chain at time t.*

Since a certificate can be revoked permanently or for a specified time period only, we say that a revoked certificate is disabled.

Definition 6. *A certificate c_1 is* ***disabled*** *at time t if there is a revocation for c_1 and t is contained in the disabling interval of c_1.*

Definition 7. *A certificate c is* ***effective*** *at time t if it is rooted at time t, t is at or after the time of issuing of c, and c is not disabled at time t.*

Definition 8. *A privilege P* ***holds*** *at time t if there is a certificate c that certifies P, c is effective at time t, and t is contained in the validity interval of c.*

2.2 Privilege Calculus

Here we present a logic program which implements the framework given above, encoding in an executable form the definitions given in the previous section. It provides a means of evaluating queries of the form

holds(P, T)

to determine whether a privilege P holds at time-point T given a database of time-stampled certificates and revocations. The program presented below can be executed as a Prolog program as it stands, once the symbol ':' is declared as an infix functor. It can also be executed in other logic programming systems to give the same results but with different computational behaviour. We use a term of the form $[T_1, T_2]$ to represent the closed-interval $[T_1, T_2]$, and a term of the form $since(T_1)$ to represent the open-ended interval of all time points later than or equal to T_1.

Rather than the *holds(P, T)* program, it is very straightforward to produce a generalisation, which is the version we present here. The 3-ary predicate

$holds(P, T, T_D)$ represents that, according to the certificates and revocations 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 certificates and revocations database, but all past states as well. This can very useful for auditing purposes, for example, or for determining the effects of retroactive certificates and revocations. (The simpler, less general version of the program, may be obtained by deleting all occurrences of the parameter T_D, and all conditions in which it appears.)

[PC 0.] T *during_interval* $[T_s, T_e] \quad \leftarrow T_s \leq T \leq T_e.$
T *during_interval since*$(T_s) \leftarrow T_s \leq T.$

[PC 1.] $holds(P, T, T_D) \leftarrow C = certifies(S, P{:}I, T_0, ID),\ T_0 \leq T_D,$
$effective(C, T, T_D),$
T *during_interval* $I.$

[PC 2.] $effective(C, T, T_D) \leftarrow C = certifies(S, Priv, T_0, ID),\ T_0 \leq T_D,$
$T_0 \leq T,$
$rooted(C, T_D),$
not disabled$(C, T, T_D).$

[PC 3.] $rooted(C, T_D) \leftarrow chain(C1, C, T_D),$
$C_1 = certifies(S, Priv, T_0, ID),\ T_0 \leq T_D,$
$sourceOfAuthority(S, P).$

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

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

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

[PC 7.] $validates(auth(S, P){:}I, C, T_D) \leftarrow C = certifies(S, P, T, ID),\ T \leq T_D,$
T *during_interval* $I.$

[PC 8.] $supports(C1, C2, T_D) \leftarrow C1 = certifies(S_1, Priv, T_1, ID_1),\ T_1 \leq T_D,$
$C2 = certifies(S_2, Priv', T_2, ID_2),\ T_2 \leq T_D,$
$validates(Priv, C2, T_D),$
not disabled$(C1, T_2, T_D).$

[PC 9.] $disabled(C, T, T_D) \leftarrow C = certifies(S, Priv, ID, T_0),$
$revokes(S, ID, I, T_1),$
$T_0 < T_1 \leq T_D,$
T *during_interval* $I.$

3 Revocation schemes in the privilege calculus

In this framework we are concerned with the revocation of *certificates*, and for reasons already explained, the resiliance dimension of [HJPW01] does not apply. We now consider how the propagation and dominance dimensions apply to the revocation schemes of ourframework.

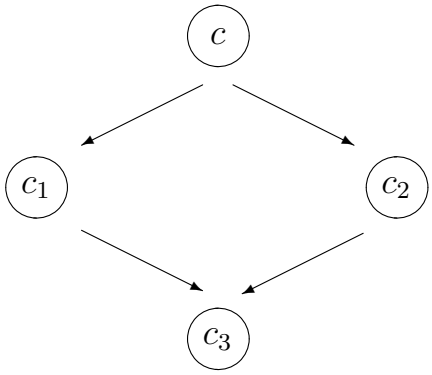

Fig. 1. c, c_1, c_2, and c_3 are certificates. An arrow between two certificates represents the support relation between the two in the direction of the arrow. For example, the arrow between c and c_1 represents that c supports c_1. A deliberate feature of the framework is that the issuer of a certificate does not have to identify the certificate that grants him the necessary authority to issue a new certificate. It is possible therefore that the same certificate can be supported by more than one certificate, as shown here. Both c_1 and c_2 could have created a privilege that made the issuing of certificate c_3 effective.

3.1 Simple Revocation

The simplest revocation scheme is the one in which an agent revokes one of his own issued certificates simply in order to withdraw the privilege given in that certificate from the time of revocation onwards. In the case where the privilege in the revoked certificate is an access-level permission, the effect of the revocation is that this particular instance of the permission does not hold at any time after the issuance of the revocation. In the case where the privilege in the revoked certificate is a management level authority the effect is that the certificate can no longer support any new certificate issued after the issuance time of the revocation. However, revocation of this certificate will not affect any other existing certificates—any certificates created by a delegation chain from the revoked certificate will continue to be effective (unless revoked directly).

This simple scheme is easily specified in the privilege calculus by a revocation with a disabling interval that starts at the revocation time and extends (open-ended) into the future: the revocation has the form $revokes(issuer, id, since(ts), ts)$.

This implies that from the time of the revocation and for any time after that, the particular instance of the privilege given in the revoked certificate is deleted.

In the diagram above, if the issuer of c revokes c at any time t after the issuance of c_1, c_2, and c_3, the revocation will not affect certificates c_1, c_2, c_3, but c cannot be used to support any new certificate issued after time t.

3.2 Propagation of revocations

Sometimes one needs to revoke a certificate in such a way that it affects the validity of some other certificates. The typical scenario is when one discovers that an agent has abused his authority. If this (perhaps fraudulent) agent is in a management role in which he has delegated privileges to others, then often one wants to delete not only his authority but also all those privileges that were delegated by him, and by his delegatees in turn.

This kind of propagation of revocations can be specified in the privilege calculus by disabling the certificate that made the fraudulent agent an authority. The certificate has to be disabled in such a way that its support for other certificates vanishes. This can be done by a revocation that has a disabling interval containing all the time points, in the past and in the future, at which that certificate supports other certificates. This is similar to: I say now that what I said in the past was not true, hence every other statement based on what I said then does not hold either. The framework we are presenting is explicitly designed to support such retrospective effects. Note that we are reasoning in two different time lines: one is the time of the certificate database, and the other is the time at which a privilege in a certificate holds.

Example: In the above picture, certificate c, issued by agent a, *directly supports* certificates c_1, issued by a_1 at time t_1, and c_2, issued by a_2 at time t_2. Further assume that a_1 and a_2 are different agents both included in the group of agents who receive the privilege given in c and that $t_1 \neq t_2$.

If at a certain time t_x, a finds out that a_1 is a fraudulent agent, the authority given to a_1 has to be deleted, immediately. But not only does the authority given to a_1 have to be deleted, the authorities delegated by a_1 also have to be deleted. Beside revoking c from time t_x, a must disable c such that its support for c_1 disappears. a can do this by first revoking c at t_x such that c cannot be used by a_1 at any time after t_x. This is a simple revocation of c at time t_x by its issuer. In order for a to delete privileges delegated by a_1, he has to revoke c again, but this time in such a way that the support relation between c and c_1 disappears. Of course, one could consider a revocation format that allows several disabling intervals to be specified in one revocation statement. One can similarly devise other kinds of 'macros' for commonly occurring patterns of revocation statements.

Note that, if c does not support c_1 any longer then automatically c_1's support for c_3 also disappears. Further, the support of c for c_2 remains the same if the disabling interval does not include t_2. Hence, the authority management structure created by a_2 remains untouched. However, for a_2 to be able to exercise,

at any time after t_x, the same authority he once received in c, a has to issue a new certificate c', at time t_x, that gives a_2, but not a_1, the same privilege that was given to him in c.

3.3 Dominance

In the framework presented so far, it is only the issuer of a certificate that has the possibility to revoke it. Relaxing this constraint will complicate the framework as well as the privilege calculus, but it is necessary if we want to support revocation schemes based on the dominance dimension. But why do we need this?

It is a deliberate feature of our framework that agent x who issues a certificate does not have to identify the certificate which grants him the necessary authority: the certificate C does not have any record of which other certificate is being invoked when certificate C is issued. This is by design. It is not realistic in our view to require that agents say "I am issuing this certificate by the authority given to me by certificate X". Agent x may not know the identifier of X, and in the case of a dormant chain, there is no such X (yet).

But now consider. Bjorn goes on holiday for the weekend and we are unable to revoke any of the certificates he issued until he gets back. The only way is to try to discover who has issued certificates to Bjorn, and then to ask each of them to revoke their certificates so Bjorn's privileges are revoked. That is clearly ridiculous. Some of the problem can be mitigated by introducing some kind of representation mechanism, allowing Bjorn to specify who can act for him in his absence. But that does not solve all the problems. It does not deal with the case where Bjorn has forgotten to appoint a representative, or done so deliberately.

The most general approach would be to decouple entirely the authority to grant privileges from the authority to revoke certificates, i.e., by introducing separate predicates $auth^+(s, p)$, which says "s has the authority to bring about p", and predicate $auth^-(s, p)$, which says "s has the authority to delete p". This would enable us to delegate each of these authorities separately. We would be able to give the authority to create a privilege to one agent and the authority to delete that privilege to a different agent.

Although this general approach would cover many interesting scenarios, and has great flexibility, we believe that it would also make the framework too powerful and too difficult to manage. Therefore we consider a less general approach, similar to that discussed under the heading of 'dominance' by [HJPW01]. Here, an agent is given the authority to revoke any certificate issued by him, and any certificate that is issued on the basis of a delegation chain stemming from one of the certificates issued by him.

The rationale is this. When an agent delegates some authority to another agent, he retains some responsibility for the actions of the delegatee. And then it seems only natural to say that the delegator should therefore retain some measure of control over how the delegated authority is used. In a certificate-based framework, this suggests the delegator should be allowed to revoke certificates issued on the basis of his original delegation acts. This also seems to be the reasoning behind the dominance mechanism discussed in [HJPW01].

The framework and the associated calculus of privileges can be modified straightforwardly. One simply replaces [PC 9.] in the privilege calculus with a more elaborate version. An example is given presently.

There are a number of outstanding points of detail to be examined, however. In particular, in a framework such as ours which supports the issuing of certificates with retrospective effects, it is easy to produce undesirable effects. For example, if a wishes to revoke a certificate C issued by b, a could simply issue a certificate granting himself the authority to create the privilege certified by C. a's certificate is not effective, but it is the start of a dormant chain supporting the certificate C. If we are not careful in specifying the dominance relation, a may be allowed the authority to revoke C in these circumstances.

One solution is to restrict attention to *rooted* chains, on the grounds that their validity rests ultimately on a source of authority whose actions need not be questioned. A second check is to compare the times of issuance of certificates in a chain, blocking the retrospective revocation of certificates through the dominance mechanism.

Such questions remain to be explored more systematically. Here we illustrate how the calculus of privileges can be modified to support the form of dominance just discussed. Investigation of other forms of dominance and their properties is a topic of current research for us.

$$\begin{aligned}[\text{PC 9x.}]\quad \mathit{disabled}(C,T,T_D) \leftarrow\ & C = \mathit{certifies}(S_1,\mathit{Priv}_1,\mathit{ID}_1,T_1),\ T_1 \le T_D,\\ & \mathit{revokes}(S_r,\mathit{ID}_1,I,T_2),\ T_2 \le T_D,\\ & T\ \mathit{during\_interval}\ I,\\ & T_1 < T_2,\\ & \mathit{dominant}(S_r,C,T_D).\end{aligned}$$

$$\begin{aligned}[\text{PC 10.}]\quad \mathit{dominant}(S_2,C,T_D) \leftarrow\ & C = \mathit{certifies}(S_1,\mathit{Priv}_1,\mathit{ID}_1,T_1),\\ & \mathit{certifies}(S_2,\mathit{Priv}_2,\mathit{ID}_2,T_2),\\ & C' = \mathit{certifies}(S_2,\mathit{Priv}_2,\mathit{ID}_2,T_2),\\ & \mathit{chain}(C',C,T_D),\\ & \mathit{rooted}(C',T_D).\end{aligned}$$

It is possible to derive equivalent but computationally more efficient formulations. However, there are other implementation issues and options to be considered. We leave detailed discussion for another paper dealing with the practical aspects of the framework.

Example: In the above picture, the issuer of c wants to delete the privilege given in c_3 which is supported by both c_1 and c_2 without deleting the privilege given in c_1 nor the privilege given in c_2. Now, what the issuer of c can do is to revoke c_3 directly without touching the validity of c_1 or c_2.

3.4 Related work

In this paper we only focus on the issue of privilege revocations in terms of certificate revocations. Hence, we will only consider the related work that deals with

the privilege revocation issues. This means that, currently, we do not consider the issues of secure and reliable revocation mechanism for distributed certificates as it is the case in PKI and PMI systems.

Most of the earlier work in revocation of privileges has focused on the consequences of revocations in terms of propagation issue. The revocation schemes discussed above are similar to those addressed in the databases literature. The *grant option* presented in [GW76] is similar to the delegations in our privilege calculus with the main distinction that in the privilege calculus one can delegate an authority to create a privilege to someone without creating the privilege for that person and/or without delegating the authority to that person to create the privilege for himself. In the framework given in [GW76] or its extension given in [Fag78], the grant capability can only be given together with the privilege itself.

In the original authorisation mechanism given in [GW76] if the same privilege is granted by the same grantor to the same subject but at different time-points then only the first one is recorded in a table and the second one is ignored. In [Fag78], the author shows that, because of how the authorisations are recorded in the authorisation mechanism given in [GW76], its revocation mechanism does not perform as it should. Hence the authorisation mechanism is extended such that similar authorisations with different time-stamps are recorded. For more details we refer the reader to [Fag78]. Note that in these authorisation mechanisms one does not revoke a particular exercise of a grant option, but the actual granted privilege, as grants themselves have no identifiers. This is different from our approach in which each delegation has its own identifier in terms of the i.d of its certificate.

A very similar approach to ours is given in [BSJ97], in which a framework for authorisations with a grant option is developed. The framework uses the grant option given in [GW76,Fag78] for further delegation of authorisations to support a decentralised management of authorisations. In this framework, negative authorisations are used for blocking positive authorisations for limited time periods. This is different from the framework given in this paper in which we use revocations with blocking time. The authors also give definitions for cascaded and non-cascaded revocations that are similar to our simple and propagation schemes for revocations.

In [BBFS97] the authors develop a framework for temporal authorisations using time intervals and temporal operators for specifying the time interval during which an authorisation is valid. In the same way a revocation may also have a time-interval that specifies the interval that the revoked authorisation becomes invalid which is similar to the disabling interval in revocations in our framework. However, this framework is also based on the same grant model as the authorisation mechanism in [GW76,Fag78], and it does not support the retrospective authorisations as in the Privilege Calculus.

4 Conclusion

We have presented an extension of our framework for updating privileges by means of authority certificates in order to provide a richer variety of revocation schemes than was originally supported. As in the original, we provide an associated calculus, encoded as a logic program, for reasoning about what privileges hold at which times given a database (or access to such a database) of time-stamped attribute certificates and revocations.

We find the classification scheme for revocation mechanisms introduced in [HJPW01], and the dimensions of resilience, propagation, and dominance identified there, extremely illuminating and helpful in structuring our own investigations. However, by focussing as we do on *mechanisms* for the creation and revocation of privileges, and on *time-dependent* effects, we find that there are a number of further distinctions that can be drawn. The concept of dominance, in particular, seems deserving of further examination, in that we have encountered a number of further points of detail that need to be resolved. We are currently undertaking a more systematic exploration of these possibilities.

A distributed privilege management system requires an architecture for secure and reliable revocation of digital certificates.

References

[BBFS97] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. Decentralized Administration for a Temporal Access Control Model. *Information Systems*, 22:223–248, 1997.

[BSJ97] E. Bertino, P. Samarati, and Sushil Jajodia. An extended authorization model for relational databases. *IEEE Transaction on Knowledge and Data Engineering*, 9(1):85–101, 1997.

[Fag78] R. Fagin. On an authorization mechanism. *ACM Tansactions on Database Systems*, 3(3):310–319, Sept 1978.

[FS02] B. Sadighi Firozabadi and M. Sergot. Revocations Schemes for Delegated Authorities. In *proceedings of IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, June 2002.

[FSB01] Babak Sadighi Firozabadi, Marek Sergot, and Olav Bandmann. Using Authority Certificates to Create Management Structures. In *Proceeding of Security Protocols, 9th International Workshop*, Cambridge, UK, April 2001. Springer Verlag. In press.

[GW76] P.P. Griffiths and B.W. Wade. An authorozation mechanism for a relational databse systems. *ACM Transactions on Databases Systems*, 1(3):242–255, 1976.

[HJPW01] Åsa Hagström, Sushil Jajodia, Francesco Parisi.Persicce, and Duninda Wijesekera. Revocation - a Classification. In *The Proceeding of the 14th Computer Security Foundation Workshop*. IEEE press, 2001.

[X.500] ITU-T Recommendation X.509: The Directory - Public-Key and Attribute Certificate Frameworks. published by International Telecommunication Union, 2000.