Revocation Schemes for Delegated Authorities *

Babak Sadighi Firozabadi
Swedish Institute of Computer Science (SICS)
Box 1263, SE-164 28 Kista, Sweden
babak@sics.se

Marek Sergot
Department of Computing
Imperial College of Science, Technology and Medicine, London
mjs@doc.ic.ac.uk

Abstract

We have an existing 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. Here we extend the framework to support a richer set of revocation schemes. The discussion of revocation 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.

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 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 [2] 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.

1.1 Revocation (deletion) Classification

In [3], revocation schemes are classified according to three dimensions—*resilience*, *propagation*, and *dominance*—which can be combined in various ways to provide several distinct categories. Although this work does not consider *revocation* of *certificates*, but rather *deletion* of *privileges* that have been granted in a delegation chain, and

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

although the representation of privileges and delegations is not the same as ours, we nevertheless find it instructive to consider how that classification might apply to 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 [3] 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 (deliberately) do not support the granting of negative privileges. 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) [1], 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. It can also be a management authority stating that the holder of the certificate has an authority to create another privilege. 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, i.e. the SoA, to subordinates. An ACRL is a list of references of attribute certificates that are no longer to be considered valid.

Our framework [2] provides a means for creating and updating authority management structures such as an AA structure in the PMI model of X.509 (2000). $perm(s, a, o)$ represents an *access level permission* which says agent s is permitted to perform action a on object o. $auth(s, p)$ represents a *management level authority* which says agent s has the authority to bring about privilege p, where 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.

In [2], 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. Here, 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 types of actions: the issuing of certificates and the revoking of certificates. Certificates are represented as:

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

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 $auth(s, q)$, 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 impact directly on the

reasoning required to determine whether a given privilege holds at a given time. Revocations are represented as:

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

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 $c_1 \ldots c_n$ 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 $c_1 \ldots c_n$ **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 such retrospective mechanisms are provided in [2].

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 issued by its issuer 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.*

3 Revocation Schemes in the Framework

In this framework we are concerned with the revocation of *certificates*, and for reasons already explained, the resilience dimension of [3] does not apply. We now consider how the propagation and dominance dimensions apply to the revocation schemes of our framework.

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. 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 framework 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 $revokes(issuer, id, since(ts), ts)$. From the time of the revocation and for all times thereafter, the particular instance of the privilege given in the revoked certificate is deleted.

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 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. Our framework 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.

Dominance As 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 , but it is necessary if we want to support revocation schemes based on the dominance dimension. But why do we need this?

In many cases, when an agent delegates some authority to another agent, he retains some responsibility for the actions of the delegatee. And then it seems natural to say that the delegator should retain some measure of control over how the delegated authority is used. In a certificate-based framework, 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 [3].

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 framework can be modified straightforwardly to support this dominance scheme. One simply replaces Definition 6 with a less restricted one as follows:

Definition 6′ *A certificate c_j is **disabled** at time t if there is a revocation r for c_j such that:*

1. *t is contained in the disabling interval of c_j, and*
2. *r is issued by the issuer of c_j, or by the issuer of c_i such that c_i directly or indirectly supports c_j in at least one **rooted** chain.*

Note that it is necessary to restrict this to rooted certificates, otherwise anybody can issue a *dormant* certificate supporting c_j and thereby gain the power to revoke c_j. Other, more complicated, dominance schemes can be defined in like manner.

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, especially concerning the concept of dominance. We are currently exploring these possibilities.

References

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

[2] B. S. Firozabadi, M. Sergot, and O. Bandmann. Using Authority Certificates to Create Management Structures. In *Proceeding of Security Protocols, 9th International Workshop*, Cambridge, UK, April 2001. Springer Verlag. In press.

[3] Å. Hagström, S. Jajodia, F. Parisi.Persicce, and D. Wijesekera. Revocation - a Classification. In *Proceeding of the 14th Computer Security Foundation Workshop*. IEEE Press, 2001.