

A Regulated Approach to Certificate Management

Victoria Ungureanu*
Rutgers University
Newark, NJ 07102
ungurean@rbs.rutgers.edu

Abstract

Traditionally, creation and revocation of certificates are performed manually, by trusted agents, under conditions that are rarely formalized. This approach to certificate management is appropriate for many current applications, where the certification or revocation of certificates is based on non-digital credentials. But it is expensive, time consuming and error-prone for the growing class of applications where credentials are digital and may be verified automatically. It is our thesis that what is needed in this situation is a mechanism that provides for the explicit formulation of certificate management policies, and for their enforcement.

In order for such an approach to be effective, the range of supported policies should not be limited to certificate management regulations alone. It is often the case that an activity cannot be fully described and understood independently of the management of the certificates it uses. Conversely, creation and revocation cannot always be specified autonomously: they may be called for as side-effects of operations unrelated to certificate management. In this paper, we will show how a control mechanism, called LGI, can be extended to support a wide range of certificate management policies as well as regulations for the various activities that use these certificates.

1 Introduction

Certificates, by which we mean digitally signed credentials of some sort, are quickly becoming the authentication method of choice. This is in part due to the advent of electronic commerce which requires means for establishing trust between parties which are physically distant from each other. And it is in part due to the fact that traditional password schemes are becoming increasingly problematic to use in large, distributed settings.

*Work supported in part by DIMACS under contract STC-91-19999 ITECC, and Information Technology and Electronic Commerce Clinic, Rutgers University

To date, several certificate frameworks [9, 6, 15] and revocation mechanisms [13, 14, 8] have been proposed and intensively studied. What interests us here is an orthogonal aspect of certificate management, which has received considerably less attention, namely *in what conditions can these mechanisms be legitimately used.*

Creation of Certificates A certificate is signed on behalf of an organization by a *certification authority* if a number of provisions are met. These conditions, which are specified by a Policy Certification Authority (PCA), generally refer to the documents needed to certify an identity, or a role [5]; and typically, PCA verification is carried out manually, off-line, by a trusted agent. Such an approach is indeed necessary if certificates are based on non-digital credentials, like birth certificates, driver's licenses, or academic diplomas. But it is time consuming, expensive and error-prone if the required credentials are digital. In this case, a more efficient approach would be to automatically verify credentials and issue the appropriate certificates.

The drawbacks of an entirely manual approach towards certificate creation can be illustrated by the following example. Consider a large, geographically distributed hospital, which buys part of the medical equipment electronically. Suppose that this hospital has the following policy regarding the purchase of goods: Only duly certified sale officers may issue purchase orders (PO); additionally, in order to be recognized as a valid order, a PO has to be signed by a designated authority¹ [10]. It stands to reason that if every little purchase is manually signed by the designated authority, the daily business of the hospital will be bottlenecked.

Revocation of Certificates Certificates might become invalid for various reasons and should be revoked. (For example, in the hospital scenario presented above, a sale officer might lose or otherwise compromise his private key, or

¹This requirement is often used in practice for the following reasons. First, it is an effective way for the client organization to ensure that its internal regulations are followed. Second, suppliers need not be aware of the structure and semantics of client certificates.

he might join a different company.) Most revocation mechanisms rely on the existence of *certificate revocation lists* (CRL) maintained by trusted servers. To revoke a certificate, the owner or another responsible authority, sends the server a signed message identifying the certificate to be revoked. Upon receipt of the message, the revocation server updates its CRL and disseminate the information [14].

But relying on the certificate owner for revocation is problematic because he can be entrusted to report an invalid certificate only in a limited number of cases. And, relying on a trusted agent, which is usually a person, is expensive, time consuming and error-prone, as we have already mentioned. But human intervention is not always needed: as we shall see, there are cases when revocation can be triggered automatically if a number of conditions—specified by the organization issuing the certificates—are met.

To illustrate the point, consider again the hospital scenario, and assume that the hospital has the following policy for entering patient orders: duly certified doctors may post patient orders; additionally, each doctor may delegate one nurse to enter orders on his/her² behalf. Since making patient orders is arguably the most important and sensitive operation performed in the hospital, one cannot entrust nurses to report when their proxy-status becomes invalid. And since each doctor in the hospital may reappoint nurses on a daily basis, requiring that each reappointment passes through a (usually central) coordinator imposes a high degree of strain on the system. An alternative solution could be to automatically revoke a proxy certificate for a doctor at the time he makes another appointment.

A New Unified Approach In this paper, we argue that creation and revocation of certificates could be viewed as any other on-line service available in a system; and access to these particular service instances, could be regulated much in the same manner as file access, surfing the web, or e-commerce. Certification and revocation policies could be, then, *formally expressed, and enforced by a generic mechanism*.

Like traditional access control policies, certificate management regulations should specify who is authorized to request what type of service. Example of such regulations, drawn from the hospital scenario, include: a request to sign a PO is authorized if issued by a duly appointed sale officer; and, a request to revoke a certificate for a doctor proxy is authorized if issued by the doctor in question. To create grounds to automate those cases that do not require human intervention, the mechanism enforcing these policies should also be able to verify digital credentials.

In order for such an approach to be effective, the range of supported policies should not, by any means, be limited to

²Henceforth, we will refer to hospital employees (sale officers, doctors, and nurses) as “he” for simplicity.

certificate management regulations alone. It is often the case that an activity cannot be fully described and understood independently of the management of the certificates it uses. A case in point, is our hospital example policy, which begs the questions: what makes a sale officer duly appointed? and who, and in what conditions could revoke his status? Conversely, creation and revocation cannot always be specified autonomously: they may be called for, as we shall see, as *side-effects* of operations unrelated to certificate management.

It is our thesis that certificate management is often intertwined with the various activities using these certificates. It follows that it should be possible that regulations regarding management of certificates *be specified as part of the control policy of the activity making use of these certificates, and be enforced by the same mechanism*. In this paper, we will show how a control mechanism, called LGI [11], can be extended to support certificate management services. We will show how LGI can be used to establish certificate management policies as well as regulations for various activities that use these certificates.

The rest of the paper is organized as follows. We start, in Section 2, with a brief description of the concept of LGI, and in Section 3, we describe extensions to LGI needed to support certificate management regulations. In Section 4 we demonstrate the proposed techniques by formulating a fairly complex hospital management policy as law under LGI. Section 5 presents related work, and we conclude in Section 6.

2 Law-Governed Interaction (LGI)—an Overview

Broadly speaking, LGI is a message-exchange mechanism that allows an *open group* of distributed agents to engage in a mode of interaction *governed* by an explicitly specified policy, called the *law* of the group. The messages thus exchanged under a given law \mathcal{L} are called \mathcal{L} -messages, and the group of agents interacting via \mathcal{L} -messages is called a *community* \mathcal{C} , or, more specifically, an \mathcal{L} -community $\mathcal{C}_{\mathcal{L}}$.

By the phrase “open group” we mean: (a) that the membership of this group (or, community) can change dynamically, and can be very large; and (b) that the members of a given community can be heterogeneous. In fact, we make here no assumptions about the structure and behavior of the agents³ that are members of a given community $\mathcal{C}_{\mathcal{L}}$, which might be software processes, written in an arbitrary languages, or human beings. All such members are treated as black boxes by LGI, which deals only with the interaction between them via \mathcal{L} -messages, making sure it con-

³Given the popular usages of the term “agent,” it is important to point out that we do not imply by it either “intelligence” nor mobility, although neither of these is being ruled out by this model.

forms to the law of the community. (Note that members of a community are not prohibited from non-LGI communication across the Internet, or from participation in other LGI-communities.)

For each agent x in a given community $C_{\mathcal{L}}$, LGI maintains, what is called, the *control-state* CS_x of this agent. These control-states, which can change dynamically subject to law \mathcal{L} , enable the law to make distinctions between agents, and to be sensitive to changes in their state. The semantics of control-states for a given community is defined by its law, and could represent such things as the role of an agent in this community, and privileges and tokens it carries. For example, under law \mathcal{HM} to be introduced in Section 4, as a formalization of our hospital policy, the term `role(doctor)` in the control-state of an agent denotes that this agent has been certified as a doctor employed in the hospital in question.

The rest of this section is organized as follows. We start in Section 2.1 with a brief discussion of the concept of law, emphasizing its local nature, and with a description of the decentralized LGI mechanism for law enforcement. Then in Section 2.2 we discuss the treatment of certificates under LGI. We do not discuss here several important aspects of LGI, including its concepts of *obligations* and of *exceptions*, the deployment of \mathcal{L} -communities, the expressive power of LGI, and its efficiency. For these issues, and for implementation details, the reader is referred to [1, 11].

2.1 Laws, and their Enforcement

Generally speaking, the law of a community \mathcal{C} is defined over a certain types of events occurring at members of \mathcal{C} , mandating the effect that any such event should have—this mandate is called the *ruling* of the law for a given event. The events subject to laws, called *regulated events*, include (among others): the *sending* and the *arrival* of an \mathcal{L} -message; and the *submission of a digital certificate*. The operations that can be included in the ruling of the law for a given regulated event are called *primitive operations*. A sample of primitive operations is presented in Figure 1.

Thus, a law \mathcal{L} can regulate the exchange of messages between members of an \mathcal{L} -community, based on the control-state of the participants; and it can mandate various side effects, such as modification of the control states of the sender and/or receiver of a message, and the emission of extra messages, for monitoring purposes, say.

On The Local Enforceability of Laws: Although the law \mathcal{L} of a community \mathcal{C} is *global* in that it governs the interaction between all members of \mathcal{C} , it is enforceable *locally* at each member of \mathcal{C} . This is due to the following properties of LGI laws:

- \mathcal{L} only regulates local events at individual agents,

$t@CS$	returns true if term t is present in the control state, and fails otherwise
$+t$	adds term t to the control state;
$-t$	removes term t from the control state;
$forward(x,m,y)$	sends message m from x to y ; triggers at y an <code>arrived(x,m,y)</code> event
$deliver(x,m,y)$	delivers the message m from x to agent y
$t@L$	returns true if term t is present in list L , and fails otherwise

Figure 1. Some primitive operations

- the ruling of \mathcal{L} for an event e at agent x depends only on e and the local control-state CS_x of x .
- The ruling of \mathcal{L} at x can mandate only local operations to be carried out at x , such as an update of CS_x , the forwarding of a message from x to some other agent, and the imposition of an obligation on x .

The fact that the same law is enforced at all agents of a community gives LGI its necessary global scope, establishing a *common* set of ground rules for all members of \mathcal{C} and providing them with the ability to trust each other, in spite of the heterogeneity of the community. And the locality of law enforcement enables LGI to scale with community size.

Distributed Law-Enforcement: Broadly speaking, the law \mathcal{L} of community \mathcal{C} is enforced by a set of trusted agents called *controllers*, that mediate the exchange of \mathcal{L} -messages between members of \mathcal{C} . Every member x of \mathcal{C} has a controller \mathcal{T}_x assigned to it (\mathcal{T} here stands for “trusted agent”) which maintains the control-state CS_x of its client x . And all these controllers, which are logically placed between the members of \mathcal{C} and the communications medium (as illustrated in Figure 2) carry the *same law* \mathcal{L} . Every exchange between a pair of agents x and y is thus mediated by *their* controllers \mathcal{T}_x and \mathcal{T}_y , so that this enforcement is inherently decentralized. However, several agents can share a single controller, if such sharing is desired.

Controllers are *generic*, and can interpret and enforce any well formed law. A controller operates as an independent process, and it may be placed on any machine, anywhere in the network. We have implemented a *controller-service*, which maintains a set of active controllers. To be effective in a widely distributed enterprise, this set of controllers need to be well dispersed geographically, so that it would be possible to find controllers that are reasonably close to their prospective clients.

On the basis for trust between members of a community:

For a member of an \mathcal{L} -community to trust its interlocutors to

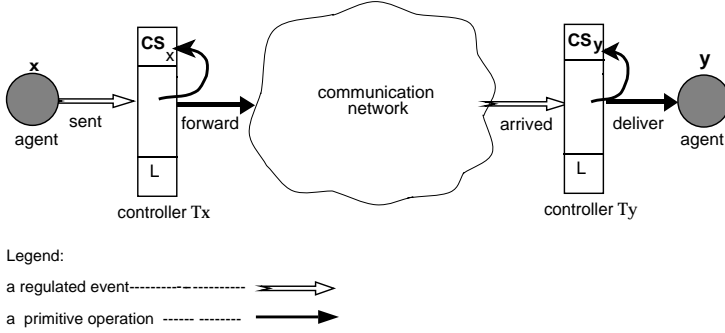


Figure 2. Enforcement of the law.

observe the same law, one needs the following assurances: (a) messages are securely transmitted over the network; (b) the exchange of \mathcal{L} -messages is mediated by controllers interpreting the *same law* \mathcal{L} ; and (c) all these controllers are *correctly implemented*. If these conditions are satisfied, then it follows that if y receives an \mathcal{L} -message from some x , this message must have been sent as an \mathcal{L} -message; in other words, that \mathcal{L} -messages cannot be forged.

Secure transmission is carried out via traditional cryptographic techniques. To ensure that a message forwarded by a controller T_x under law \mathcal{L} would be handled by another controller T_y operating under the *same law*, T_x appends a one-way hash [12] H of law \mathcal{L} to the message it forwards to T_y . T_y would accept this as a valid \mathcal{L} -message under \mathcal{L} if and only if H is identical to the hash of its own law.

As to the correctness of controllers, we assume here that every \mathcal{L} -community is willing to trust the controllers certified by a given certification authority (CA), which is specified by law \mathcal{L} . And, every pair of interacting controllers must first authenticate each other by means of certificates signed by this CA.

2.2 The Treatment of Certificates under LGI

Under LGI, *all agents are made equal* at the time they join an \mathcal{L} -community. This is because the control-state of all new members is identical—and control-states provide the only means for a law to make distinctions between agents. We now explain how an agent can acquire extra privileges by submitting appropriate certificates.

The submission by an agent x , operating under law \mathcal{L} , of a certificate $Cert$ to its controller, has the following effect: An attempt is made to confirm that $Cert$ is a valid certificate, duly signed by an authority that is acceptable to law \mathcal{L} , i.e., an authority that is represented by one of the *authority-clauses* in the preamble to the law (See Figure 3 for an example). If this attempt is successful⁴, then

⁴If the certificate is found invalid then an *exception-event* is triggered.

a *certified-event* is triggered. This event, which is one of the *regulated-events* under LGI, has as its argument the following representation of the submitted certificate:

```
[issuer(I), subject(S),
attributes(A)].
```

Here I and S are internal representations of the public-keys of the CA that issued this certificate, and of its subject, respectively; and A is what is being certified about the subject. Structurally, A is a list of `attribute(value)` terms. For example, the attributes of a certificate might be the list `[name(johnDoe), role(doctor)]`, asserting that the name of the subject in question is JohnDoe and his role in this community is a doctor. Additional components of the attributes field include the expiration time of the certificate, the URL of the server that maintains CRLs for this type of certificates, a certificate id (used to identify it in CRLs), etc. (Currently we support SPKI format of certificates [6]).

What happens when the *certified event* is triggered depends, of course, on the law. In the case of law \mathcal{HM} of Figure 3, for example, when a doctor-certificate is presented, triggering the *certified event*, first the law attempts to verify if the certificate has not been revoked and also to monitor any status changes in the future. This is done by sending a `monitorStatus(Cert, Freq)` request to a *certification authority proxy* (CAP), where `Freq` is the monitoring frequency. A CAP serves as an interface between communities under LGI and the certification infrastructure they use. (This is just one facet of the functionality supported by a CAP. For a full description of CAPs, as well as more examples dealing with expiration and revocation of certificates, the interested reader is referred to [1].)

A CAP is trusted to perform the requested checking using the most recent information available (either via an online server or by examining the CRL), and to respond via an \mathcal{L} -message `status(Stat, Cert)`, where `Stat` is the status of certificate `Cert`. And after its first status report, the CAP will monitor the status of `Cert`, with the specified frequency. A CAP is trusted to report immediately to the issuer of this request any change of the status of the monitored certificate. The effect of such a report is, of course, determined by the law under which the requester operates, as we shall see in the example presented in Section 4.

3 Establishing Certificate-Management Policies

To support the formulation of certificate-management policies via laws we introduce into LGI two trusted agents: a *certifier* and a *registrar*. Their functionalities essentially mirror the roles performed by a certifying authority, and by a revocation server, respectively. More precisely, a certifier,

is a server entitled to sign credentials on behalf of a community. And a registrar acts as a repository for certificates, and publishes information related to their revocation.

The certifier and the registrar differ, however, from their traditional counterparts, in an essential respect: they serve only \mathcal{L} -regulated requests, where \mathcal{L} is the law of the organization which established them. As we shall see, this feature makes possible to control, in a formal manner, who, and in what conditions, is allowed to create and revoke credentials. We will now briefly describe the types of requests, represented by \mathcal{L} -messages (for any law \mathcal{L}), recognized by these trusted agents.

Certifier Certifiers are designed to respond to requests to sign arbitrary statements. This type of request is denoted by messages of the form

```
certify(Stmnt),
```

where `Stmnt` is the statement to be certified.

A certifier, `C`, responds to a `certify` request, with a message

```
certified(Stmnt,Cert),
```

where `Cert` is a digital certificate for `Stmnt`, signed with `C`'s private key.

Registrar As we have mentioned above, a registrar is a server designed to maintain certificates, and to periodically publish certificate revocation lists (CRLs). In the current implementation, beside the digital certificate per se, a registrar also records its LGI internal format. For now, a registrar recognizes the following set of messages:

- `publish(Cert)`—which denotes a request to maintain and publish certificate `Cert`;
- `revoke(Stmnt)`—denotes a request to revoke the certificate that matches exactly `Stmnt`. This is done by checking `Stmnt`, given as a Prolog-list, against the LGI internal format of the maintained certificates. If a match is found, the corresponding digital certificate is immediately revoked.
- `revoke*(Stmnt)`—denotes a request to revoke all certificates that contain `Stmnt`. It is a generalization of `revoke`, which allows that all certificates containing a given set of attributes be nullified in one step. For example, a request `revoke*([requester(johnDoe)])` will nullify *all* credentials that have (among others) the attribute `requester(johnDoe)`.
- `test&revoke(Stmnt)`—denotes a request to report the current status of the certificate matching

`Stmnt`, if one is found, and to revoke it. The testing and the revocation are performed in one atomic step.

The use of all but the last request will be illustrated below by showing how the hospital management policy, introduced informally in Section 1, can be implemented as a law under LGI. As for `test&revoke`, we mention that it was introduced in order to support policies where the rights of an agent change dynamically as a result of its actions, like, for example, Chinese Wall [4]. Due to space constraints, we do not present here a formal example of its use.

4 A Case Study: The Hospital Management Law

We illustrate here the nature of certificate management regulations by presenting a thorough description of the rules governing hospital activities which were introduced schematically in Section 1. This policy, which we call HM, for “hospital management”, has the following five points dealing both with certificate management aspects (points 1 to 3) and with hospital activity, per se (points 4 and 5).

1. *The role of the various participants in this activity must be validated via digital certificates issued by a certification authority called `admin`. A revocation server called `pub` is entrusted by the hospital with maintaining and revoking certificates.*
2. *Certificate management is regulated as follows. An agent, duly certified as a system administrator, is authorized to create or revoke any certificate. An agent, duly certified as a doctor, is authorized to request the creation of a proxy-doctor certificate; and he is authorized to request the revocation of any certificate issued on his behalf. Finally, a sale officer is authorized to request the certification of a purchase order.*
3. *Once a submitted certificate is verified, its validity is to be monitored as follows: certificates attesting sale officer and doctor privileges are to be checked for validity every hour, while the other certificates should be checked every 30 seconds. (This difference in frequencies reflects different expectation about the stability of the certificates.) Whenever a certificate is revoked, its corresponding privileges must be nullified immediately.*
4. *Only duly certified doctors and their duly appointed proxies may post patient orders.*
5. *Only duly certified sale officers may issue purchase orders (PO); for a PO to be valid, it has to be signed by `admin`.*

We will now show how the HM-policy can be established under LGI by law \mathcal{HM} displayed, in its entirety, in Figures 3 through 6.

4.1 Establishing Certificate-based Privileges

The fragment of law \mathcal{HM} dealing with treatment of certificates is displayed in Figure 3. This figure has two parts, specifying the *preamble* to the law, and its rules. Each rule is followed by a comment (in italic), which, together with the following discussion should be understandable even for a reader not well versed in the LGI language for writing laws.

The preamble to this law has several clauses: First, there are two *authority* clauses, which define the authorities acceptable to this community. Each authority clause provides the public-key of an authority, and assign it a local name—`admin` and `pub`, in this case—to be used within this law. As we shall see, `admin` is trusted to serve as a certifier and `pub` as a registrar. Second, there are three *alias* clauses specifying addresses for three distinguished agents: `admin`, `pub` and `cap`. As we will see later, `cap` is the certification authority proxy entrusted by this community⁵ with verifying certificates and monitoring their status. Finally, there is an *initialCS* clause that defines the initial control-state of agents in this community, which is empty in this case.

Our discussion of this law-fragment is organized as follows: We start with how an agent that adopted this law can claim a role—of system administrator, a security officer, or a doctor—by presenting a specified type of certificate, signed by the stipulated authority. Second, we show how these credentials are verified and how the role mentioned in the certificate is recorded in the control state of the agent via a role term such as `role(doctor)`.

By Rule $\mathcal{R}1$, when an agent, `X`, presents a self certificate, `Cert` issued by `admin`, a `monitorStatus` request is sent to `cap`. The privileges implied by this certificate are conferred to bearer, `X`, only after `cap` asserts that `Cert` is valid. By Rule $\mathcal{R}3$ this is done as follows. If there is an attribute in `Cert` which attests that the agent can play role `R`, then this attribute is recorded into control state by a term `role(R)`; and if there is an attribute `N`, certifying the identity of the agent in question, then it is similarly recorded via a term `id(N)`.

Rule $\mathcal{R}4$ deals with the case when a certificate has been revoked after an agent `X` has been granted privileges on its base. (As mentioned above, such changes are monitored by `cap`, and reported as soon they occur.) In this case the corresponding `role` and `id` terms are removed from the control state of `X`, thus effectively preventing `X`, as we shall see, from exercising privileges in the community.

⁵ Assumed here to be Bellevue.

4.2 Creation and Revocation of Credentials by System Administrators

We have shown in the previous section how certificates can be used to establish terms like `role` and `id` into the control state of their bearers. We shall show now, how these terms, entitle agents to special privileges; in particular, we will present a law fragment, displayed in Figure 4, which allows agents having the term `role(sys_admin)`, to create and revoke arbitrary credentials.

By Rule $\mathcal{R}5$, a request addressed to `admin`, to sign a statement on behalf of an agent `SA`, is delivered to `admin` only if `SA` has `role(sys_admin)` in its control state. Given the way this term was established, this ensures that only properly authorized system administrators may issue this type of requests.

The certifier responds with a `certified(Stmnt, Cert)` where `Cert` represents the signed `Stmnt`. By Rule $\mathcal{R}6$, such a message is forwarded to the requester, without further ado. Once a certificate arrives at the requester, `SA`, the certificate is immediately delivered to two agents: (1) to `SA`, to notify him that the certificate has been issued; and (2) to `pub`, which will publish it (see Rule $\mathcal{R}7$).

Finally, by Rule $\mathcal{R}8$, only an agent having a `role(sys_admin)` may ask `pub` to revoke any previously published certificate.

4.3 Making Patient Orders

We have seen in the last subsection that according to the hospital policy, system administrators may issue credentials for hospital employees. Examples of such credentials include certificates that allow their bearers to assume a doctor, nurse or sale officer role. We will show now how these certificates may be used to regulate hospital activity. We start by presenting how doctor certificates are used to control releasing of patient orders and appointment of proxy-doctors; and we conclude the example by showing in the next paragraph how purchasing may be regulated by means of sale officer certificates.

The sending, by `X`, of a request to `admin` to certify `N` as a proxy-doctor is handled by Rule $\mathcal{R}9$, displayed in Figure 5, as follows. If `X` does not have the term `role(doctor)` in his control state, then this message is simply ignored. Otherwise the following operations are carried out: (a) if, `X` already appointed a nurse, `N'`, to act as his proxy, then a request to revoke the proxy certificate of `N'` is automatically delivered to `pub` on behalf of `X`; (b) a `certify` message is sent to `admin` to request a proxy certificate for `N`; and (c) a term `proxy(N)` is added to the control state of `X` to record that `N` acts now as a proxy for `X`.

Finally, by Rule $\mathcal{R}10$, a patient order is delivered to a

```

Preamble:
  authority(admin,publicKey1).
  authority(pub,publicKey2).
  alias(admin,"admin@bellevue.com").
  alias(pub,"pub@bellevue.com").
  alias(cap,"cap@bellevue.com").
  initialCS([]).

R1. certified([issuer(admin),subject(Self),attributes(A)]) :-
    if (role(doctor)@A or role(sale_officer)@A)
        then Freq=[1,hour]
        else Freq=[30,s]
    do(deliver(Self,monitorStatus(Cert,Freq),cap)).

    If an agent presents a self-certificate, issued by admin, cap is asked to check the status of the presented certificate, and to monitor its status.

R2. sent(cap,status(S,Cert),X) :- do(forward).

    Only cap can send status messages.

R3. arrived(cap,status(valid,Cert),X) :- attributes(A)@Cert,
    if id(N)@A then do(+id(N))
    if role(R)@A then do(+role(R)).

    If the certificate is valid, the privileges implied by it are granted, i.e. N, the id of the bearer, and R, the role that X may hold, are recorded into the control state by terms id(N) and role(R).

R4. arrived(cap,status(revoked,Cert),X) :- attributes(A)@Cert,
    if (role(R)@A and role(R)@CS) then do(-role(R))
    if (id(N)@A and id(N)@CS) then do(-id(N)).

    If a certificate is revoked, then the privileges established on its base are removed immediately.

```

Figure 3. Fragment of law HM dealing with establishing certificate-based privileges

server only if sent by authorized doctors and their proxies, i.e. by agents that have either term `role(doctor)` or `role(proxy_doctor)` in their control state.

4.4 Issuing Purchase Orders

The purchasing activity of the hospital is regulated by Rules $\mathcal{R}11$ and $\mathcal{R}12$, displayed in Figure 6, as follows. First, when an agent, properly authorized to act as a sale officer, issues a purchase order, PO, for a supplier, S, a request to sign PO is automatically sent to admin. When the certifier responds, the ruling mandates that the purchase offer together with its certificate are to be finally delivered to the supplier (Rule $\mathcal{R}12$).

Discussion We note here that under this implementation of HM-policy certain certificate management operations occur without a pro-active cause. For example, the certification of a purchase order occurs as a side-effect of a valid request to issue a PO. Similarly, the revocation of a proxy for a doctor is automatically requested on behalf of the doctor in question when he appoints another nurse.

5 Related Work

The concept of a unified, automated control of both credentials and application is not entirely new. Capabilities, which are traditionally used for controlling file access in centralized systems, are automatically generated and distributed when a file has been created, and are nullified when a file has been removed. However, policies regarding management of capabilities were hardwired in the reference monitor, while here we insist on making them explicit. This is important given the multitude of applications using certificates, and therefore the potential diversity of this type of policies.

Blaze, Lacy and Feigenbaum developed a formalism called PolicyMaker [3, 2], where, like in LGI, the agent privileges are determined by the certificates it carries, and are specified explicitly. Unlike LGI, however, in PolicyMaker one cannot create or revoke certificates dynamically as a side-effect of the ruling. As such a number of policies, including the HM-policy presented here, cannot be stated in their formalism.

The only other framework, we are aware of, that con-

```

R5. sent(SA,issueCredential(Stmnt),admin) :- role(sys_admin)@CS
      do(deliver(SA,certify(Stmnt),admin).

  Only an agent that has the term role(sys_admin) in its control state may send issueCredential requests.

R6. sent(admin,certified(Stmnt,Cert),SA) :- do(forward).

  Certificates issued by admin are forwarded without ado to the destination.

R7. arrived(admin,certified(Stmnt,Cert),SA) :-
      do(deliver(X,publish(Cert),pub)), do(deliver).

  Once a certificate arrives, the receiver, SA, is notified of the outcome, and a request to publish Cert is automatically made on its behalf.

R8. sent(SA,revokeCredential(Stmnt),pub) :- role(sys_admin)@CS,
      do(deliver(SA,revoke(Stmnt),registrar).

  An agent holding the role of sys_admin may ask for the revocation of any certificate.

```

Figure 4. Fragment of law \mathcal{HM} dealing with creation and revocation of credentials by system administrators.

```

R9. sent(X,appointProxy([role(proxy_doctor),id(N),key(K)]),admin) :-
      role(doctor)@CS, id(D)@CS,
      if proxy(N')@CS then
        (do(deliver(X,revoke*([requester(D)]),pub)), do(-proxy(N')))
      do(deliver(D,certify([role(proxy_doctor),id(N),key(K),requester(D)],admin),
        do(+proxy(N))).

  An agent, that has role(doctor) established by an authorized issuer, may send an appointProxy message. Note, that the certificate will contain, in addition to the initial statement, the term requester(D). Additionally, if X has previously appointed N' to act as his proxy, then this certificate is automatically revoked.

R10. sent(D,order(O),S) :-
      role(doctor)@CS or role(proxy_doctor)@CS,
      do(deliver).

  Only an agent certified to be a doctor or a proxy for a doctor may send an order.

```

Figure 5. Fragment of law \mathcal{HM} dealing with patient orders

siders the issue of regulated certificate management is Oasis [7]. There are several distinctions between Oasis and the mechanism presented here. First, the system deals only with a special type of certificates, namely certificates establishing roles. Second, while Oasis supports policies regulating certificate creation of comparable sophistication to ours, the range of revocation policies is far narrower. In their framework, policies deal only with propagation of revoked certificates; Oasis does not attempt to regulate the (more general) case of who is allowed to revoke a certificate, and in what conditions. Finally, certificate management is viewed in isolation of the activity using the certificates.

6 Conclusion

Management of creation and revocation of digital certificates is one of the most complex aspects of distributed access-control. Usually certificate management is performed manually, by trusted agents, and it is rarely formalized. We have argued that such an approach is expensive, error-prone and might impose a high degree of strain on a system. To deal with these problems we have proposed in this paper a mechanism for the specification of a wide range of certificate management policies, and for their enforcement.

Of course, it is not possible to automate entirely certificate management. For example, in the hospital scenario, issuing or revoking a doctor or a sale officer certificate requires human intervention. It is worth noting, however, that

```

R11. sent(SO, issue(PO), S) :-
    role(sale_officer)@CS,
    do(+ pending(PO, S)),
    do(deliver(SR, certify(PO), admin)).

```

An agent may issue a PO only if it has a term `role(sale_officer)` in its control state. If this is the case then the following actions will be taken: (1) a term `pending(PO, S)` is added to the control state to record the name of supplier S, and (2) a request to certify PO is sent to admin.

```

R12. arrived(admin, certified(PO, Cert), SO) :-
    pending(PO, S)@CS, do(-pending(PO, S)),
    do(deliver(SO, [PO, Cert], S)).

```

When admin responds by sending Cert, a certificate for PO, the purchase order together with the certificate are delivered to supplier S, the intended destination.

Figure 6. Fragment of law \mathcal{HM} dealing with purchase orders

these operations occur far less frequently than issuing a PO, or appointing a nurse as a doctor-proxy. It follows then, that a potentially small base of manually issued certificates can be used to establish a considerably larger number of digital credentials. Moreover, as various business activities are performed more and more electronically, it is likely that there will be a significant trend towards expressing and maintaining credentials digitally. This, in turn, might make the manual process entirely obsolete.

References

- [1] X. Ao, N. Minsky, and V. Ungureanu. Formal treatment of certificate revocation under communal access control. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 116–129, Oakland, California, May 2001.
- [2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, 1603, 1999.
- [3] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1996.
- [4] D. Brewer and M. Nash. The Chinese Wall security policy. In *Proceedings of the IEEE Symposium in Security and Privacy*, pages 206–211, Oakland, California, 1989. IEEE Computer Society.
- [5] C. Ellison. Generalized certificates, 1996.
- [6] C. Ellison. The nature of a usable PKI. *Computer Networks*, (31):823–830, November 1999.
- [7] R.J. Hayton, J.M. Bacon, and K. Moody. Access control in an open distributed environment. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1998.
- [8] J. Iliadis, Spinellis. D., D. Gritzalis, B. Preneel, and S. Katsikas. Evaluating certificate status information mechanisms. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 1–8, November 2000.
- [9] S Kent. Internet privacy enhanced mail. *Communications of the ACM*, August 1993.
- [10] H. Ludwig, L. O’Connor, and S. Kramer. Miera - method for inter-enterprise role-based authorization. In *Proceedings of the Conference on Electronic Commerce and Web Technologies*, pages 133 – 144, 2000.
- [11] N.H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *TOSEM, ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, July 2000.
- [12] B. Schneier. *Applied Cryptography*. John Wiley and Sons, 1996.
- [13] S. Stubblebine. Recent-secure authentication: Enforcing revocation in distributed systems. In *Proceedings of the 1995 IEEE Symposium on Research in Security and Privacy, Oakland*, pages 224–234, May 1995.
- [14] R. Wright, P. Lincoln, and J. Millen. Efficient fault-tolerant certificate revocation. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, November 2000.
- [15] R. Zimmermann, P. *The Official PGP User’s Guide*. Boston:MIT Press, 1995.