

Flexible Treatment of Certificate Revocation Under Communal Access Control

Abstract

The conventional approach to distributed access-control (AC) tends to be *server-centric*. Under this approach, each server establishes its own policy regarding the use of its resources and services by its clients. The choice of this policy, and its implementation, are generally considered the prerogative of each individual server.

This approach to access-control may be appropriate for many current client-server applications, where the server is an autonomous agent, in complete charge of its resources. But it is not suitable for the growing class of applications where a group of servers, and sometimes their clients, belong to a single enterprise, and are subject to the enterprise-wide policy governing them all. One may not be able to entrust such an enterprise-wide policy to the individual servers, for two reasons: First, it is hard to ensure that an heterogeneous set of servers implement exactly the same policy. Second, as we will argue, an AC policy can have aspects that cannot, in principle, be implemented by servers alone.

It is our thesis that what is needed in this situation is a concept of *communal policy* that governs the interaction between the members of a distributed community of agents involved in some common activity, along with a mechanism that provides for the explicit formulation of such policies, and for their scalable enforcement. This paper focuses on the communal treatment of expiration and revocation of the digital certificates used for the authentication of the identity and roles of members of the community.

1 Introduction

The conventional approach to distributed access-control (AC) tends to be *server-centric*. Under this approach, each server establishes its own policy regarding the use of its resources and services by its clients. The choice of this policy, and its implementation, are generally considered the prerogative of each individual server. In fact, the server's AC policy is often embedded, completely or partially, in its code. This is particularly true for those aspects of the policy that cannot be expressed via the traditional access-matrix model, such as separation of duty constraints, auditing requirements, and other provisions of the so called "commercial policies" [?]. Another critical aspects of AC policies, which is generally relegated to the code of individual servers, is the treatment of stale or revoked certificates, where certificates are used for the authentication of the identity of clients or of their roles.

Such a server-centric approach to access-control may be appropriate for many current client-server applications, where the server is an autonomous agent in complete charge of its resources. But this approach is not suitable for the growing class of applications where a group of servers, and sometimes their clients, belong to a single enterprise, and are subject to the enterprise-wide policy governing them all.

Consider, for example, a large geographically distributed hospital complex, whose patient records are handled by several *heterogeneous* R-servers ("R," for records), maintained

by various departments, labs, and outpatient units distributed throughout the hospital. The hospital is likely to have a general policy governing the entry, update, and retrieval of patient records, independently of the R-server that happens to maintain them. Such a policy would typically specify how participants such as doctors, nurses and the R-servers themselves ought to be authenticated, perhaps via certain kinds of certificates; it would spell out the privileges to be granted to each such participant; and it would specify the effect that the revocation of a certificate should have on the privileges of the participant previously authenticated by it.

One cannot entrust such a hospital-wide policy to the individual servers, for two reasons: First, it is hard to ensure that an heterogeneous set of servers implement exactly the same policy, particularly if the policy is to be implicit in the code of each server. Second, as we shall see later, an AC policy can have aspects that cannot, in principle, be implemented by servers alone.

It is our thesis that what is needed in this situation is a concept of *communal policy* that governs the interaction between the members of a distributed community of agents involved in some common activity, along with a mechanism for formulating and enforcing such policies. Such a mechanism needs to satisfy the following requirements:

1. Communal policies should be made *explicit*, and be *enforced*.
2. The enforcement mechanism for communal policies should not require central control.

The first of these requirements is critical for any heterogeneous community, whose members cannot all be relied on to observe one overarching policy voluntarily. The second requirement above is important for large and geographically distributed systems, where any central enforcer of policies could become a performance bottleneck, and a dangerous single point of failure.

This concept of communal AC policy has been originally introduced in [?], but without taking into account the possible revocation of the digital certificates used for the authentication of the identity and roles of members of a community. The subject of communal certificate-revocation policy is taken up in this paper.

The rest of this paper is organized as follows: We start, in Section 2, with the nature of communal policies, illustrating this concept with a patient-records example; Section 3 extends this concept by taking into account the possible expiration and revocation of digital certificates. In Section 4 we provide an overview of the concept of law-governed interaction (LGI), which we use for the implementation of communal policies. In Sections 5 and 6 we discuss the treatment of expiration and revocation of certificates, respectively, and we formalize the example-policies introduced in Sections 2 and 3. We make some concluding remarks in Section 7.

2 The Concept of Communal Policy

We illustrate here the nature of communal policies by focusing on one aspect of the treatment of patient records in hospitals: the entering of doctor's orders into such records. Doctor's orders, or simply "orders", are, arguably, the most sensitive aspect of patient record, as they may directly affect the treatment, and thus the health and life, of patients. Suppose, then, that our hospital has the following three-point policy regarding the entering of doctor's orders; we call this the PR policy, for "patient records".

1. *The role of the various participants in this activity must be validated via digital certificates issued by specific certification authorities (CAs), as follows:*
 - (a) *Servers that maintain patient records (R-servers) must be certified as such by a CA called `admin` (for the hospital administration).*
 - (b) *Doctors must be certified by two CAs: (1) a CA we call `board` (for the medical board of the state) certifying that the person in question has an MD degree, and (2) by the CA `admin` (mentioned above), certifying that he or she¹ has doctor’s privileges in this hospital.*
2. *Orders must be posted only on duly certified R-servers, and only by duly certified doctors.*
3. *A copy of every posted order must be sent to the designated audit-trail server.*

This is a *role-based* policy, where the roles of doctors and of R-servers are to be validated by digital certificates. This by itself is not new (role-based access control (RBAC) is quite common now [?, ?], as is the use of certificates for validating roles [?, ?].) What interests us here about this policy is its communal nature. That is, the requirement that this policy should govern *all* doctor-orders within the hospital.

A communal policy of this kind cannot be implemented by the servers alone, even if all servers can be trusted to enforce the same policy. This is for two reasons. First, R-servers cannot ensure that orders are sent only to certified R-servers, as required by Point 2 of this policy. They obviously cannot prevent a doctor from sending his order elsewhere, to some agent that has not been certified as an R-server. This, clearly, requires a degree of control over the actions of the doctors. Second, as argued in [?], a communal policy might be concerned not only with client/server interactions, but also with interactions between clients. For example, one may want to allow doctors to appoint certain nurses as their surrogates, delegating to them the right to make orders. Such an appointment may be carried out by an exchange between a doctor and a nurse, not involving any of the R-servers—which are, therefore, in no position to regulate it.

Of course, the concept of communal AC policy is not really new. Policies under the original access-matrix model [?], were implicitly communal—governing all “subjects” and “objects” in a given central operating-system. Such a policy was to be enforced by means of a centralized *security kernel* that mediates all access requests. A centralized enforcement of this kind is quite effective within a single host, but not for a large distributed system, where it could become a bottleneck, and a dangerous single point of failure—as we already pointed out. There is clearly a need for a decentralized mechanism for the enforcement of such policies in distributed systems. Before we get to such a mechanism we address an important aspect of communal policies: their treatment of certificate-revocation.

3 Communal Certificate-Revocation Policies

Public key certification is an essential element of access control over the Internet [?]. Unfortunately, unlike diamonds, certificates *are not forever*. They tend to get “stale” in time, so they often contain a time stamp, and an explicit validity period. Moreover, certificates might be revoked, for various reasons, even within their validity period.

¹Henceforth, we will refer to doctors as “he,” for simplicity.

It is, therefore, not sufficient to specify which privileges to bestow on a principal that presents a given set of certificates—as we have done in our PR policy above. One needs to have a policy that deals with the possibility that any of these certificates will become invalid, *after* such privileges have been granted. And such a policy must be communal, just as the other provisions of the PR policy. We outline here some of the issues involved in formulating such a policy, and we will amend our example policy PR accordingly, for the sake of illustration. We start with the case of certificate expiration, and then deal with the more complex case of explicit revocation.

Dealing with certificate expiration: Suppose that the AC policy at hand confers a bestows p on any agent x that presents a set C of certificates. The question is: what should be done when one of these certificates expires? Here are some of the possible answers to this question.

- The privilege p bestowed on x is nullified immediately.
- x is given a grace period during which it retains its original privilege.
- x is given a grace period, as above; but his privileges are reduced in some way during this period.
- If x delegated his privilege p to other agents (say, if a doctor appointed a nurse as his surrogate²), then the delegated privileges must be nullified along with p .

To show how some of these options can be applied in practice we now consider the following amendment of our example policy PR:

4. *When the certificate used to authenticate a given server as an R-server expires, the server's ability to receive orders should be immediately nullified.*
5. *When any one of the certificates used to authenticate a given agent d as a doctor expires, d should be notified of this, and be given a grace period of a specified duration, within which he can present a fresh certificate. If a fresh certificate is not presented by the end of this grace period, then d should lose his ability to send orders.*

Thus amended, our example policy is denoted by PR'. Its formalization under the mechanism to be proposed here is presented in Section 5.

Dealing with certificate revocation: When a certificate is known to have been revoked one faces the same types of choices presented by the expiration of that certificate. The new issue here is *finding out when a certificate has been revoked*.

The dissemination of information about revoked certificates is part of the job of certification authorities, and of the supporting infrastructure. Various such dissemination mechanisms have been proposed [?]. The most common mechanism of this kind, used under the X.509 standard [?], employs *certificate revocation lists* (CRLs), which are published periodically. Another mechanism, which can be used along with CRLs, allows for online checking of the status of a given certificate.

It is up to the policy that governs the use of certificates to determine their desired recency [?], and thus the frequency in which their validity is to be verified with respect to

²The policy governing such a delegation should also be communal, as shown in [?].

the available revocation information. For example, one may require that the validity of a given type of certificates be verified when it is first presented, and periodically, with a given frequency, thereafter. Alternatively, one may require that the validity of a certificate be verified whenever the privilege implied by it is employed.

There is another issue to consider. The checking of the validity of a certificate involves communication with a CA, or with a CRL repository. Such communication may be delayed indefinitely due to network congestion, or due to some denial of service attack. Consequently the revocation policy should deal with the case when the state of a certificate is simply *unknown*. To illustrate these issues we now introduce the following amendment of our example policy PR’:

6. *Once a submitted certificate is verified, its validity is to be monitored as follows: certificates issued by **admin** are to be checked for validity every day, while certificates issued by **board** should be checked every 30 days. (This difference in frequencies reflects different expectation about the stability of the certificates issued by the two authorities.)*
7. *Whenever a certificate is revoked, its corresponding privileges must be nullified immediately. (Note that we have chosen here, somewhat arbitrarily, a harsher measure than the one employed, under Point 5 above, for expiration of certificates.)*
8. *When the status of a certificate becomes unknown, the privileges corresponding to it should be retained, temporarily, for a specified uncertainty period. If the status of this certificate remains unknown by the end of this period, then the corresponding privileges are to be nullified.*

Thus amended, this policy is denoted by PR”. Its formalization under the mechanism to be proposed here is presented in Section 6.

A note about related work is appropriate here. Revocation policies of comparable sophistication to the above can probably be implemented under the Oasis system [?]. But while providing a very powerful certification infrastructure, Oasis does not support communal policies. That is, it provides no assurances that a given community of agents is governed by a designated AC policy.

4 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

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.

are treated as black boxes by LGI, which deals only with the interaction between them via \mathcal{L} -messages, making sure it conforms 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 $\mathcal{C}_{\mathcal{L}}$, LGI maintains, what is called, the *control-state* \mathcal{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 dynamic changes in their state. The semantics of control-states for a given community is defined by its law, could represent such things as the role of an agent in this community, and privileges and tokens it carries. For example, under law \mathcal{PR}' to be introduced in Section 5, as a formalization of our example PR' 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.

We now elaborate on several aspects of LGI, focusing on (a) its concept of law, (b) its mechanism for law enforcement, and (c) its treatment of digital certificates. Due to lack of space, we do not discuss here several important aspects of LGI, including the *interoperability* between communities, and the treatment of *exceptions*. Nor do we discuss here the expressive power of LGI, its implementation, and its efficiency. For these issues, and for a more complete presentation of the rest of LGI, the reader is referred to [?, ?, ?].

4.1 The Concept of Law

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; the *coming due of an obligation* previously imposed on a given object; and the submission of a digital certificate (more about the latter two kinds of events, later). The operations that can be included in the ruling of the law for a given regulated event are called *primitive operations*. They include, operations on the control-state of the agent where the event occurred (called, the “home agent”); operations on messages, such as `forward` and `deliver`; and the imposition of an obligation on the home agent.

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 of the message-exchange, 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,
- the ruling of \mathcal{L} for an event e at agent x depends only on e and the local control-state \mathcal{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 \mathcal{CS}_x , the forwarding of a message from x to some other agent, and the imposition of an obligation on x .

Operations on the control-state	
$t@CS$	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;
Operations on messages	
$forward(x,m,y)$	sends message m from x to y ; triggers at y an $arrived(x,m,y)$ event
$deliver(x,m,y)$	delivers the message m from x to agent y
Miscellaneous	
$t@L$	returns true if term t is present in list L , and fails otherwise
$imposeObligation(type,dt)$	causes the triggering of an $obligationDue(type)$ event dt seconds later.

Figure 1: Some primitive operations in LGI

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.

On the Structure and Formulation of Laws: Abstractly speaking, the law of a community is a function that returns a *ruling* for any possible regulated event that might occur at any one of its members. The ruling returned by the law is a possibly empty sequence of primitive operations, which is to be carried out locally at the location of the event from which the ruling was derived (called the *home* of the event). (By default, an empty ruling implies that the event in question has no consequences—such an event is effectively ignored.)

Concretely, the law is defined by means of a Prolog-like program⁴ L which, when presented with a goal e , representing a regulated-event at a given agent x , is evaluated in the context of the control-state of this agent, producing the list of primitive-operations representing the ruling of the law for this event. In addition to the standard types of Prolog goals, the body of a rule may contain two distinguished types of goals that have special roles to play in the interpretation of the law. These are the *sensor-goals*, which allow the law to “sense” the control-state of the home agent, and the *do-goals* that contribute to the ruling of the law. A *sensor-goal* has the form $t@CS$, where t is any Prolog term. It attempts to unify t with each term in the control-state of the home agent. A *do-goal* has the form $do(p)$, where p is one of the above mentioned primitive-operations. It appends the term p to the ruling of the law. A sample of primitive operations is presented in Figure 1.

The Concept of Enforced Obligation: Informally speaking, an obligation under LGI is a kind of *motive force*. Once an obligation is imposed on an agent—generally, as part of

⁴Note, however, that Prolog is incidental to this model, and can, in principle, be replaced by a different, possibly weaker, language; a restricted version of Prolog is being used here.

the ruling of the law for some event at it—it ensures that a certain action (called *sanction*) is carried out at this agent, at a specified time in the future, when the obligation is said to *come due*, and provided that certain conditions on the control state of the agent are satisfied at that time. The circumstances under which an agent may incur an obligation, the treatment of pending obligations, and the nature of the sanctions, are all governed by the law of the group.

Specifically, an obligation can be imposed on a given agent x at time t_0 by the execution at x of a primitive operation

`imposeObligation(oType,dt),`

where dt is the time period, after which the obligation is to come due, and $oType$ —the *obligation type*—is a term that identifies this obligation (not necessarily in a unique way). The main effect of this operation is that unless the specified obligation is *repealed* before its due time $t=t_0+dt$, the *regulated event*

`obligationDue(oType)`

would occur at agent x at time t . The occurrence of this event would cause the controller to carry out the ruling of the law for this event; this ruling is, thus, the *sanction* for this obligation. Note that a pending obligation incurred by agent x can be *repealed* before its due time by means of the primitive operation

`repealObligation(oType)`

carried out at x , as part of a ruling of some event. (This operation actually repeals *all* pending obligations of type $oType$).

For example, under law \mathcal{PR}' , when a server s submits a certificate authenticating it as an R-server, an obligation `expired(server,valid)` is imposed on s , to come due at the expiration time of this certificate. When this obligation comes due, it will cause s to lose its ability to receive orders. For example, under law \mathcal{PR}' , when an agent d submits a certificate authenticating it as a doctor, an obligation `expired(doctor,valid)` is imposed on d , to come due at the expiration time of this certificate. When this obligation comes due, it will cause d to lose its ability to send orders.

Note that there is a significant difference between this concept, and the concept of obligation under *deontic logic* [?], used for the specification of normative systems. The obligations of deontic logic allow one to reason about *what an agent must do*, but they provide no means for ensuring that what needs to be done will actually be done [?].

4.2 The Law-Enforcement Mechanism

We start with an observations about the term “enforcement,” as used here. We do not propose to coerce any agent to exchange \mathcal{L} -messages under any given law \mathcal{L} , just as we cannot coerce doctors to issue their orders via a computer rather than via pen and paper. The role of enforcement here is merely to ensure that *any exchange of \mathcal{L} -messages, once undertaken, conforms to law \mathcal{L}* . More specifically, our enforcement mechanism is designed to ensure the following properties: (a) the sending and receiving of \mathcal{L} -messages conforms to law \mathcal{L} ; and (b) a message received under law \mathcal{L} has been sent under the same law (i.e., it is not possible to forge \mathcal{L} -messages).

Since we do not compel anybody to operate under any particular law, or to use LGI, for that matter, how can we be sure that all doctor orders in a given hospital are entered

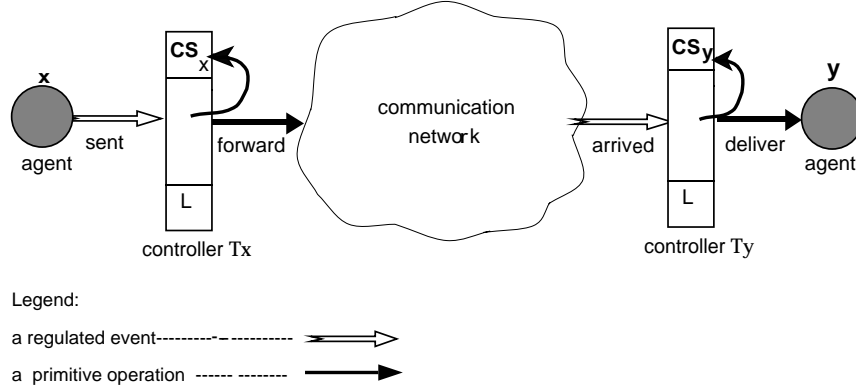


Figure 2: Enforcement of the law

under law \mathcal{PR}' ? The answer is that an agent may be *effectively compelled* to exchange \mathcal{L} -messages, if he needs to use services provided only under this law, or to interact with agents operating under it. For instance, if the servers for patient records accepts only \mathcal{PR}' -messages, then anybody needing their services would be compelled to send \mathcal{PR}' -messages to them. Conversely, if doctors make their orders via \mathcal{PR}' -messages, servers would be compelled to accept such messages, if they are to receive any orders.

This is not an *absolute* assurance for universal use of law \mathcal{PR}' for doctor orders in a given hospital. Indeed a group of doctors may decide to use some rogue server, not operating under \mathcal{PR}' , for making their orders. But this would be a blatant defiance of the probably explicit hospital rule that doctor's orders must be made only under law \mathcal{PR}' . This is far less likely to happen than some subtle (or even not subtle) change in the policy adopted by some server, under the server-centric approach to AC.

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 \mathcal{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. Although several agents can share a single controller, if such sharing is desired. (The efficiency of this mechanism, and its scalability, are discussed in [?].)

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 members of an \mathcal{L} -community to trust its interlocutors to observe the same law, one needs the following assurances: (a) that the exchange of \mathcal{L} -messages is mediated by controllers interpreting

the *same law* \mathcal{L} ; and (b) that all these controllers are *correctly implemented*. If these two 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.

To ensure that a message forwarded by a controller \mathcal{T}_x under law \mathcal{L} would be handled by another controller \mathcal{T}_y operating under the *same* law, \mathcal{T}_x appends a one-way hash [?] H of law \mathcal{L} to the message it forwards to \mathcal{T}_y . \mathcal{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.

With respect to the correctness of the controllers, if an agent is not concerned with malicious violations, then it can trust a controller provided by our controller-naming service, or a controller provided by the operating system – just like we often trust various standard services on the Internet, such as TCP/IP protocols. When malicious violations are a concern, however, the validity of controllers and of the host on which they operate needs to be certified. In this case, the controller-naming service needs to operate as a *certification authority* for controllers. Furthermore, messages sent across the network must be digitally signed by the sending controller, and the signature must be verified by the receiving controller, allowing the two controllers to trust each other. Such secure inter-controller interaction has been implemented in Moses ([?]).

4.3 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, thus becoming *more equal than others* (with apologies to George Orwell), 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 successful⁵, then 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 [?]).

What happens when the **certified** event is triggered depends, of course, on the law. In the case of law \mathcal{PR}' of Figure 3, for example, the following would happen when a doctor-certificate is presented, triggering the certified event: (a) the term **role(doctor)** is set in

⁵If the the certificate is found invalid then an *exception-event* is triggered.

the control-state of the agent in question, and (b) an obligation is imposed to deal with the eventual expiration of this certificate.

5 Establishing Communal Certificate-Expiration Policies

We demonstrate here the communal treatment of certificate-expiration by formalizing our example PR' policy into a law \mathcal{PR}' under LGI, which is displayed, in its entirety, 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 discussion⁶ 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 certification authorities acceptable to this community. Each authority clause provides the public-key of a certification authority, and assign it a local name—“admin” and “board”, in this case—to be used within this law. Second, an **initialCS** clause that defines the initial control-state of all agents in this community, which is empty in this case. Finally, there is a **directory** clause defining the address of the specific agent. `auditor@bellevue.com`, and assigns it a local name—“auditor”. As we will see later, this agent would be used as a server that maintains the audit trail for doctor orders for the hospital at hand (assumed here to be Bellevue).

Our discussion of this law is organized as follows: We start with how an agent that adopted this law can claim a role—of an MD, a doctor, or a server—by presenting a specified type of certificate, signed by a specified authority; and how this role is recorded in the CS of the agent via a role term such as `role(server)`. Such terms serve as a kind of *seal* that authenticate the role of an agent for its interactions with other members of the \mathcal{PR}' -community. For example, the term `role(server)` serves, under this law, as a *seal* of a valid R-server, which allows agents with this term to receive orders. Second, we show how orders are sent by qualified doctors and how they are received by qualified servers. Finally, we discuss what happens if the certificate previously used to authenticate the role of an agent expires.

By Rule $\mathcal{R}1$ and $\mathcal{R}2$, an agent that presents a certificate issued by `board`, with the attribute `role(md)`, or a certificate issued by `admin`, with attribute `role(doctor)` or `role(server)`, would get the corresponding term `role(md)`, `role(doctor)` or `role(server)` in its CS. The name `N` of the agent in question is authenticated in a similar manner, and is recorded via a term `name(N)`. By Rule $\mathcal{R}2$, whenever a certificate is presented, an obligation will also be imposed to deal with its expiration. Furthermore, if the privilege associated with this certificate is in its grace period, then the corresponding obligation is repealed.

By Rule $\mathcal{R}3$, an agent can issue orders only if he has both `role(md)` and `role(doctor)` terms in its control state. Such an order must be included in a message of the form `order(text(O),myName(N))`, where `O` is the order itself, and `N` is the name of the doctor sending this message, as certified previously and recorded in the control-state of the doctor. If the receiver of an order message is a valid R-server, then by Rule $\mathcal{R}4$ a message `audit(M)`, where `M` is the text of the order, is sent to an agent whose address is `auditor@bellevue.com`—as defined by the **directory** clause before. Otherwise an `uncertifiedServer` message will be sent back to the sender of the order (Rule $\mathcal{R}5$).

⁶Due to space limitations, our discussion of this and the following law is perhaps more terse than it should be.

Rule $\mathcal{R}6$ deals with the expiration of server-certificates. Whenever the server-certificate expires, the term `role(server)` will be removed from the control-state of that agent, thus ensuring that this agent can no longer receive any orders. By Rule $\mathcal{R}7$, once an MD-certificate or a doctor-certificate expires, the agent in question gets a grace period of one day, along with a warning message.

Finally, by Rule $\mathcal{R}1$, while in its grace period, an agent can introduce a new and fresher certificate, which will get him out of the grace period and into a normal operating mode. But if he fails to present such a certificate in time, he will have his privileges removed when the grace period expires (by Rule $\mathcal{R}8$).

6 Establishing Communal Certificate-Revocation Policies

To support the formulation of certificate-revocation policies via laws we introduce the concept of *certification authority proxy* (CAP), which serves as an interface between communities under LGI and the certification infrastructure they use. A CAP is designed to respond to two kinds of requests, represented by the following \mathcal{L} -messages (for any law \mathcal{L}):

- `checkStatus(Cert)`—a request to check the current status of the certificate `Cert`, given in its LGI format introduced in Section 4.3.
- `monitorStatus(Cert,Freq)`—a request to check the current status of `Cert`, and to monitor any status changes in the future, according to the specified frequency. (The monitoring will continue until the certificate is no longer valid, or until the CAP is instructed, in a manner not spelled out here, to stop the monitoring.)

The 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(S,Cert),`

where `S` is the status of certificate `Cert` that can have the following values: `valid`, `revoked` or `unknown`. The `unknown` status occurs when the CAP cannot get to the appropriate server, due to communication problems, or because the server is unavailable at the time.

In the case of `monitorStatus(Cert,Freq)` request: 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 certificate transition diagram is presented in Figure 4). 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 below.

Finally, we note that a single community may use several different CAPs, which may be geographically dispersed, and may be built to handle different types of certification infrastructure that use different revocation mechanisms.

Dealing with Certificate Revocation Under the PR” Policy: To illustrate how revocation policies can be implemented under LGI, we revise here our previous law, creating law \mathcal{PR}'' that represents policy PR” introduced in Section 3. This revision is presented in Figure 5, which replaces Rule $\mathcal{R}1$ from Figure 3 with Rule $\mathcal{R}1'$, and adds Rules $\mathcal{R}9$ – $\mathcal{R}15$. Also, this revision contains a `directory` clause that defines `cap@bellevue.com` as the address of the CAP to be used by this law—its internal name is to be `cap`.

Preamble:

```
authority(admin,publicKey1).
authority(board,publicKey2).
initialCS([]).
directory(auditor, "auditor@bellevue.com").
```

$\mathcal{R}1.$ `certified([issuer(I),subject(Self),attributes(A)]) :-
 role(R)@A, (I==board,R==md)|(I==admin,(R==doctor|R==server)),
 name(N)@A, expirationTime(T1)@A, clock(T)@CS,
 T2=T1-T, T2 >0, setRole(R,N,T2).`

Claiming or reclaiming the role of MD, doctor or server via certificate issued by CAs(called board and admin).

$\mathcal{R}2.$ `setRole(R,N,T) :-
 if (role(R)@CS)
 then do(repealObligation(expired(R,S)))
 else (do(+role(R)), do(+name(N))),
 do(imposeObligation(expired(R,valid),T)).`

When setting a role, an obligation is imposed to deal with its expiration, if there is already an obligation associated with the role, it is repelled.

$\mathcal{R}3.$ `sent(D,order(text(D),myName(N)),S) :-
 role(doctor)@CS, role(md)@CS, name(N)@CS, do(forward).`

Only an agent that has role(doctor) and role(md) in its control-state can issue the orders.

$\mathcal{R}4.$ `arrived(X,M,S) :- M=order(-,-),
 if role(server)@CS
 then (do(deliver), do(deliver(S,audit(M),auditor)))
 else do(forward(S,uncertifiedServer(M),X)).`

The arrival of a message carrying a doctor's order.

$\mathcal{R}5.$ `arrived(S,uncertifiedServer(M),X) :- do(deliver).`

A "uncertifiedServer" message is delivered without further ado.

$\mathcal{R}6.$ `obligationDue(expired(server,valid)) :-
 do(-role(server)), name(N)@CS, do(-name(N)),
 do(deliver(Self,warning(serverCertExpired),Self)).`

A warning will be sent to the server and its server privilege will be removed when the server certificate expires.

$\mathcal{R}7.$ `obligationDue(expired(R,valid)) :- (R==md)|(R==doctor),
 do(imposeObligation(expired(R,grace),[1, day])),
 do(deliver(Self,warning(certExpired(R)),Self)).`

When the doctor-certificate or MD-certificate expires, the law will degrade the agent to be in its one day long grace period. The agent will also receive a warning message.

$\mathcal{R}8.$ `obligationDue(expired(R,grace)) :- (R==md)|(R==doctor),
 do(-role(R)), name(N)@CS, do(-name(N)),
 do(deliver(Self,warning(graceExpired(R)),Self)).`

If the grace period of the privilege expires, we will remove corresponding role(md) or role(doctor) term from its control-state, and thus remove the corresponding privilege.

Figure 3: Law \mathcal{PR}'

Figure 4: The certificate state-transition diagram

Under this law, when an agent x presents a certificate \mathbf{Cert} ⁷, a `monitorStatus` request is sent to `cap` (Rules $\mathcal{R}1'$ and $\mathcal{R}9$). The privileges implied by this certificate are conferred on x only after it gets a status report from `cap`, asserting that this certificate is valid (see Rule $\mathcal{R}12$).

The rest of the law deals with the case when the status of a certificate changes *after* an agent x has been granted privileges—i.e. after a term `role(R)` has been added to x 's control state. As mentioned above, such changes are monitored by the `cap` and reported as soon as they occur.

If the certificate is revoked, then, by Rule $\mathcal{R}13$ the corresponding `role` term is removed from the control state of x immediately, thus effectively preventing x from exercising privileges in this community. A warning is also sent to the affected agent.

The last requirement of this policy stipulates that x should maintain its role R , for a given *uncertainty* period (six hours, in this case say) if the status of the corresponding certificate \mathbf{Cert} , becomes unknown. This is achieved in the following manner: First, by Rule $\mathcal{R}14$, when `cap` reports that the status of \mathbf{Cert} has become `unknown` an obligation `expired(R,uncertain)` is set to fire at x after six hours—the duration of the uncertainty period. If during this time frame, `cap` reports that the status of \mathbf{Cert} changed to `valid`, then the `expired` obligation is repealed (Rule $\mathcal{R}12$). Otherwise, when the obligation fires, the term `role(R)` is removed from the control state of x , thus nullifying its privileges (Rule $\mathcal{R}15$).

7 Conclusion

The treatment of expiration and revocation of digital certificates is one of the more complex aspects of distributed access-control. Such treatment is usually embedded in the code of servers, and is rarely formalized. But when dealing with a communal policy that governs several heterogeneous servers, as well as their clients, it is necessary for this treatment to be formalized as part of the access control policy at hand, and to be enforced. This is what we have done in this paper.

What is not discussed in this paper, due to lack of space, is the effect of revocation of certificates on the delegation of privileges within a community, such as when a doctor delegates his right to make orders to a nurse. This aspect of communal policies, and a discussion of the efficiency of our treatment of certificates and of their revocation, is left for a forthcoming paper.

⁷Variable \mathbf{Cert} is automatically bound to the internal translated form of the certificate presented by the agent.

Preamble:

directory(cap, "cap@bellevue.com").

$\mathcal{R}1'$. certified([issuer(board),subject(Self),attributes(A)]) :-
role(md)@A, do(forward(Self,monitorStatus(Cert,[30,day]),cap)).

After receiving MD certificate signed by board, the cap is asked to check the status of the certificate presented, and to monitor its status every 30 days.

$\mathcal{R}9$. certified([issuer(admin),subject(Self),attributes(A)]) :-
role(R)@A, (R==doctor|R==server),
do(forward(Self,monitorStatus(Cert,[1,day]),cap)).

After receiving a certificate from admin, the cap is asked to check the status of the certificate presented, and to monitor its status every day.

$\mathcal{R}10$. arrived(X,monitorStatus(C,F),cap) :- do(deliver).

Only the cap can receive monitorStatus messages.

$\mathcal{R}11$. sent(cap,status(S,C),X) :- do(forward).

Only the cap can send status messages.

$\mathcal{R}12$. arrived(cap,status(valid,C),X) :-
attributes(A)@C, role(R)@A, name(N)@A, expirationTime(T1)@A,
clock(T)@CS, T2=T1-T, T2 >0,
if (obligation(expired(R,uncertain))@CS) then
(do(-role(R)),do(-name(N)),
do(repealObligation(expired(R,uncertain))))),
setRole(R,N,T2).

If the certificate is valid, the privileges implied by it are granted. Furthermore, if the status of that certificate changes into valid from the previous unknown status, then the corresponding obligation is repealed.

$\mathcal{R}13$. arrived(cap,status(revoked,C),X) :-
attributes(A)@C, role(R)@A, do(-role(R)),
do(deliver(Self,warning(certRevoked(R)),Self)).

If the certificate is revoked, then the privileges emanating from it are removed immediately.

$\mathcal{R}14$. arrived(cap,status(unknown,C),X) :-
attributes(A)@C, role(R)@A,
do(imposeObligation(expired(R,uncertain),[6,hour])).

If the status of the certificate changes into unknown from previous valid status, then the agent is given an uncertainty period of six hours.

$\mathcal{R}15$. obligationDue(expired(R,uncertain)) :-
role(R)@CS, do(-role(R)),
do(deliver(Self,warning(uncertainExpired(R)),Self)).

When the uncertainty period expires, the corresponding role, and thus privilege, is removed.

Figure 5: \mathcal{PR}'' —a revision of the patient record law \mathcal{PR}'