Regulated Delegation,  
and its Role in Distributed Access Control  

Xuhui Ao and Naftaly H. Minsky  
{ao,minsly}@cs.rutgers.edu, Department of Computer Science  
Rutgers University, Piscataway, NJ 08854  

November 5, 2003  

Abstract  

Certificate-based delegation (CBD) is a critical element in distributed access control, providing it with flexibility and scalability. But despite its elegance and effectiveness, CBD has inherent limitations that restrict its application domain. These limitations include, among others: lack of support for non-monotonic policies, such as separation of duties; the inability to support the transfer of privileges, where the delegator loses the privilege it delegates; and the lack of support for quotas, i.e., restrictions on the number of time a given privilege can be exercised.

This paper describes an approach to the distributed delegation problem, which shares much of the flexibility and scalability of CBD, but is not encumbered by limitations such as above. This approach is based on the distributed control mechanism called law-governed interaction (LGI), which employs a distributed set of reference monitors.

The resulting treatment of distributed delegation is not a replacement for CBD, but is complementary to it. First, because their range of suitable application domains is complementary. Second, because LGI itself is based on CBD, and uses it explicitly, when appropriate.

1 Introduction  

One of the main difficulties in conducting access control over large, heterogeneous, distributed systems, is gaining reliable information about one’s interlocutors, and knowing what can they be trusted for. The currently leading answer to this difficulty is the so called trust management approach to authorization, introduced and developed under several projects, such as: SPKI/SDSI [6], KeyNote [5], and Delegation Logic [8]. They all rely on what has come to be known as “certificate-based delegation,” (CBD, for short). This powerful approach has been described succinctly in [8], as follows:
In the "trust-management" approach to distributed authorization, a "requester" submits a request, possibly supported by a set of "credentials" issued by other parties, to an "authorizer," who controls the requested resources. The authorizer then decides whether to authorize this request by answering the "proof-of-compliance" question: "Do these credentials prove that a request complies with my local policy?".

The credentials mentioned in this description include the so called delegation certificates—which are particularly effective when SPKI/SDSI is employed, identifying principals (delegators and delegatees) with their public keys.

Despite the elegance of this approach, and the flexibility and scalability it provides (which have been discussed in a superb study by Aura [3]), CBD has inherent limitations that restrict its application domain. These limitations—which are well recognized by the proponents of CBD [3]—can be attributed to two, related, aspects of this approach: (1) its server-centric nature; and (2) its lack of control over the process of delegation. We now discuss these aspects, and their implications, in detail.

The server-centric nature of CBD: Certificate-based delegation is “server-centric” in the sense that the meaning of a delegation certificate depends on the policy of the server that controls the requested resource—the “authorizer” in the above description of trust management—and which is the final arbiter of the rights that should be provided to anybody submitting a given set of certificates.

Such an approach to access-control is quite natural for many current client-server applications, where the server is an autonomous agent in complete charge of its resources. But, as argued in [2], this approach is not suitable for peer-to-peer communities; or for the growing class of applications where a group of servers, and their clients, belong to a single enterprise, and are subject to a mandatory, enterprise-wide, policy governing them all. (We will see an example of such a mandatory policy in the following section).

Moreover, the fact that the meaning of a delegation certificates is relative to the policy of the intended server, and does not have any intrinsic meaning, is problematic even in the case of a truly autonomous server that employs its own policy. The problem is that the holder of a given certificate cannot be sure of its implications, without knowing the details of the policy employed by the server. And it is hard to be sure about this policy, in particular, because policies are often implicit in the code of the server, and because the server is free to change its policy at will.

The lack of control over the process of delegation: It is one of the basic principles of CBD that it assumes no control over the process of delegation; and imposes no dependency of the issuance of delegation-certificates, on the state of the delegator, and of the delegatee.
All that matters, according to CBD, is the bundle of certificates submitted to the authorizer, which it evaluates according to its own policy.

This freewheeling delegation process provides CBD with great flexibility, and scalability. Unfortunately, the integrity of some systems requires that certain constraints be satisfied by the delegation process itself. And it is not always possible for the authorizer (i.e., the server), that gets a chain of certificates, to determine if the required constraints have been satisfied during the formation of this chain.

A case in point is the requirement that a certain right would be delegated only by transfer; that is, that the delegator would give up its own right, when delegating it to somebody else. It is, of course, impossible for the authorizer to determine whether or not the delegator did give up his own right. Delegation by transfer is, therefore, not supported by CBD. Other limitations of CBD that are due to its lack of control over the delegation process—already discussed in [3]—include: lack of support for non-monotonic policies, such as separation of duties; and the inability to support a quota for the number of times that a given right can be used. In the following section we will illustrate these limitations via an example.

Moreover, the lack of control over the delegation process may be problematic even if the policy at hand allows the authorizer to determine the right of the requester purely on the basis of the bundle of certificates it presents. This is for two reasons: First, the need to verify the validity of large number of certificates for every service request could be overly time consuming. Second, as we will demonstrate in the following section, the authorizer may be in no position, and, indeed, have no right, to evaluate the delegation process that provides the requester with the right for the service in question. As a simple example, note that when accepting a dollar bill as payment for a service, we do not examine the sequence of “delegations” that transferred this bill from the mint that produces it, all the way to us. We trust, instead, that this bill has always been transferred, from one hand to another, and never copied. No such trust is available under CBD.

Of course, as pointed out in [3], all these limitations of CBD can be eliminated by employing a centralized trusted computing base (TCB) to operate as a reference monitor that supervises all accesses, and all delegations of access rights, while they are carried out. But by doing this we will lose the flexibility and scalability of certificate-based delegation.

The contribution of this paper: Following an analysis of the nature of the limitations of CBD, and of their underlying reasons, we will point to a different approach to the distributed delegation problem, which is not encumbered by the limitations of CBD, but share much of its flexibility and scalability. This approach is based the distributed control mechanism called law-governed interaction (LGI), which employs a distributed set of reference monitors—collectively forming a decentralized trusted computing base, or DTCB—all interpreting the same communal policy, called the law of this community. Among other things, such a law can be written to provide a firm semantics to privileges, and it can regulate their
delegation.

The resulting treatment of distributed delegation is not a replacement for CBD, but is complementary to it. First, because their range of suitable application domains is complementary. Second, because LGI itself is based on CBD, and uses it explicitly, when appropriate.

The rest of this paper is organized as follows: We start with an example that would help us demonstrate the above mentioned limitations of CBD. In Section 3 we provide a very brief introduction to LGI, more details about which can be found in the Appendix, and in [2, 1]. In Section 4 we show how the example policy introduced in Section 2 is implemented under LGI; and in Section 5 we reflect on the nature of delegation under LGI, and on its complementarity with CBD. We conclude in Section 6.

2 On the Limitations of Certificate-Based Delegation

We will attempt here to demonstrate the limitations of CBD via an example. We will show later how this example can be formalized and enforced by a LGI, with some use of CBD.

2.1 An Example

Consider a community $C$ of distributed agents (people, and software components working on their behalf) belonging to some enterprise $E$. These agents are partitioned into two disjoint groups, called management and staff. We will be interested in the purchasing activity by the members of the staff (also to be called buyers), carried out by sending purchase orders (POs) to various vendors (which are not part of enterprise $E$). And we will focus on the following policy $PP$ (for “purchasing policy”) that specifies the manner in which the purchasing activity is to be supervised by certain members of the management group.

Policy $PP$: This policy involves three managerial roles: chief, supervisor and auditor (see Figure 1). Their authority with respect to the buyers, and with respect to each other, is specified below:

1. There is a single, fixed, chief in community $C$. It can appoint two different members of the management group to the roles of supervisor and auditor.

2. The chief can supply the supervisor with a purchasing budget, which the supervisor has the right to distribute among buyers—without the right to use this budget for issuing its own POs.\(^2\)

\(^1\)Since an agent, in this context, may be either a person or a software component, we will refer to agents by “it”.

\(^2\)We leave the authority of the auditor unspecified, for simplicity.
3. Each buyer is allowed to issue arbitrary POs, provided that their cumulative cost does not exceed the buyer’s budget.

4. The supervisor is allowed to delegate its supervisory role to another member of the management group, subject to the following constraints:
   
   (a) This delegation must be by transfer, like the passing of a leader’s baton, so that the community would have no more than one supervisor at a time.
   
   (b) When delegating its supervisory role, one has to transfer its remaining purchasing budget to the new supervisor.
   
   (c) The role of supervisor should never be delegated to an agent playing the role of an auditor. (This is an example of what is known as separation of duties constraint).
   
   (d) The chief must be notified of every transfer of the supervisor role.

2.2 Discussion

The most notable aspect of this policy, as compared to the concept of policy under CBD, is that it is communal. That is, it is not the policy of any particular agent, with respect to the use of its resources, but it governs the interaction of various agents in community $C$. Indeed, the main purpose of this policy is to regulate the process of delegation in this community, in order to ensure that certain global, i.e. communal, properties be always satisfied. These include the following properties: (a) the total cost of purchase orders issued by members of this community never exceeds the budget provided by the chief; (b) there can never be
more that one supervisor in the community, at the same time; and (c) the same agent cannot play the roles of supervisor and auditor at the same time.

One may attempt to cast this communal policy into the CBD framework, as follows: We will have each vendor adopt a policy of accepting a PO only if it is accompanied with a “proper” chain of delegation certificates. Namely, a chain starting with the certificate issued by the chief appointing his first supervisor; continuing with the sequence of certificates that delegated the supervisory role, from one agent to another; and ending with the certificate signed by the last of these supervisors, providing the requester with its budget. But this attempt has several serious problems:

- It is hard to ensure that all heterogeneous vendors adopt the same, server-centric policy.
- By examining the set of certificates presented to it, a vendor would not be able to ascertain that most of the requirements of our $PP$ policy have been satisfied. In particular, and basically for the reasons explained in [3], the vendor would not be able to ascertain: (a) that there is only one supervisor in the system; (b) that the supervisor does not play the role of auditor as well; and (c) that the requesting buyer still has sufficient funds in its budget, and did not use these funds by POs sent to other vendors.
- The vendors, which do not belong to enterprise $E$, might not be in a position, or have the right, to see the relevant set of delegation certificates, or to check if they satisfied policy $PP$ (even if they were able to do so). The reason is that policy $PP$ may be considered a private matter of the enterprise, which it may want to change occasionally without having to inform the various outside vendors it might trade with. Moreover, even if the policy itself is to be known to the vendors, the specific agents involved in a given delegation chain might be considered confidential by the enterprise.

We will see later why these difficulties do not mar our LGI-based implementation of this policy.

3 Law-Governed Interaction (LGI)–a Brief Overview

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 an $\mathcal{L}$-community, denoted by $\mathcal{C}_\mathcal{L}$, or simply by $\mathcal{C}$.

We provide here only a very brief overview of this mechanism, attempting to make the rest of the paper understandable for people not previously familiar with LGI. More
information about LGI, particularly about the structure of its laws, its law enforcement mechanism, its deployment, and its treatment of certificates, is provided in the appendix. For additional information the reader is referred to [9, 2, 1].

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 agents that are members of a given community \( C \), which might be software processes, written in arbitrary languages, or human beings. Both the clients and the servers of the traditional distributed system terminology are viewed here as agents. All the members are treated as black boxes by LGI, which deals only with the interaction between them via \( L \)-messages, ensuring conformance 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 \( L \)-community, LGI maintains, what is called, the control-state \( CS_x \) of this agent. These control-states, which can change dynamically, subject to law \( L \), 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.

**The nature of LGI laws:** An LGI law is defined over a certain types of events occurring at members of a community \( C \) subject to it, mandating the effect that any such event should have. Such a mandate is called the *ruling* of the law for the given event. The events subject to laws, called *regulated events*, include (among others): the *adoption* of the law by the
agent, the sending and the arrival of an L-message. The operations that can be included in the ruling for a given regulated event, called primitive operations, are all local with respect to the agent in which the event occurred (called, the “home agent”). They include, operations on the control-state of the home agent and operations on messages, such as forward, deliver and release. Our middleware currently provides two languages for writing laws: Java, and a somewhat restricted version of Prolog. We employ prolog in this paper. In this case, the law is defined by means of a Prolog-like program 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 cs, producing the list of primitive-operations r representing the ruling of the law for this event.

**Distributed Law-Enforcement:** Broadly speaking, the law L of community C is enforced by a set of trusted agents called controllers, that mediate the exchange of L-messages between members of C. Every member x of C has a controller Tx assigned to it (T here stands for “trusted agent”) which maintains the control-state CSx of its client x. And all these controllers, which are logically placed between the members of C and the communications medium (as illustrated in Figure 2) carry the same law L. Every exchange between a pair of agents x and y is thus mediated by their controllers Tx and Ty, 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 [9].)

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.

Finally, we point out that under the current implementation of LGI, a controller takes about 100 microseconds for every evaluation of a law of the size of law PP introduced in the following section.

**4 A Case Study**

We now show how the purchasing policy PP, described in Section 2, can be established, in its entirety, via LGI. We start by formalizing policy PP into an LGI law PP. And we will try to explain this law, attempting to make it understandable even to people not previously familiar with LGI. Then, in Section 4.2, we will amend this law slightly, enhancing the trustworthiness of the POs sent under this law, to the vendors that do not operate under it.
4.1 Law $PP$

Law $PP$ is displayed in Figures 3 and 4. Like any other LGI law, it has two parts: called the *Preamble*, and the *body*.

The preamble of $PP$ has several clauses. The *law* clause indicate its name, and the public key of a certificate authority (CA) that is to be used for certifying the controllers interpreting this law. (Note that every law can specify its own CA.) The *authority* clause specifies the public key of a CA—called here ‘‘admin’’—whose certification would be accepted, under this law, for the authentication of agents as employees of certain types of our example enterprise $E$. Finally, the *alias* clause specifies the address of the distinguished agent chief. The body of this law is a list of rules, each of which is followed by informal comments, in italics.

We discuss this law by explaining its treatment of the various activities it regulates, as follows: the *adoption* of this law, an act that makes an agent into a member of the $PP$-community; the appointment of a supervisor, and of auditors, by the chief; the delegation, by transfer, of the supervisory role; the assignment of budgets to buyers, by the supervisor; and the sending of POs to vendors.

(a) Adopting law $PP$:

For an agent $x$ to adopt law $PP$, thus joining the $PP$-community it needs choose a generic controller, supply it with the law $PP$ under which it wishes to operate, and supply it a certificate issued by the enterprise administrator–admin, with the attribute $\text{type(management)}$ or $\text{type(staff)}$. By Rule $R1$, this would cause either the term $\text{type(management)}$ or $\text{type(staff)}$ to be added to its control state. Due to the provisions of law $PP$, this term would create a major distinction between these two kinds of employees.

We note here that any agent that can supply a required type of certificate can join this community, at any time. And that there is no central authority that regulates the community membership, or that is involved, in any way, in the act of joining it. This is generally true under LGI.

(b) Appointment of the supervisor and of auditors:

By Rule $R2$, the chief can appoint the initial supervisor by sending the $\text{appoint-supervisor}(B)$ message where $B$ is an arbitrary initial purchasing budget; he can do that only if he did not already appoint one, as evidenced by the $\text{sAppointed}$ term. The appointment will take place upon the arrival of this message, according to Rule $R3$, and only if the appointee is a management agent and doesn’t hold the role of auditor (thus observing the separation of duties constraint$^3$ of this policy). Otherwise, an exception message will be forwarded to the chief.

$^3$We note that this provision does not prevent the same person from adopting law $PP$ twice, and then assuming the roles of supervisor and auditor at the same time. This issue can be handled under LGI, at the cost of some complexity, as shown in [1].
Preamble:

law(name(pp),ca(publicKey1)).
authority(admin,publicKey2).
alias(chief, chief@enterprise.com).

R1. adopted(Arg) :- if not(Arg==certificate(issuer(admin),subject(Self),
                          attributes([type(T)])), \(T\in\{management,staff\}\))
    then do(qit) else do(+type(T)).

Only the employee with the management or staff certificate issued by the admin can adopt this law.

R2. sent(chief, appoint-supervisor(B),X)
    :- not(sAppointed@CS), do(+sAppointed), do(forward).

Only the chief can appoint the initial supervisor, giving it an arbitrary initial budget B—but only if currently there is no supervisor.

R3. arrived(chief, appoint-supervisor(B),X)
    :- if (role(auditor)@CS|not(type(management)@CS))
       then do(forward(X, exception(failed-delegation(B)), chief))
       else do(+role(supervisor)), do(+budget(B)), do(deliver).

Acceptance of the appointment to be the supervisor.

R4. sent(chief, appoint-auditor,X) :- do(forward).

Only the chief can appoint the auditors.

R5. arrived(chief, appoint-auditor,X)
    :- if (role(supervisor)@CS|not(type(management)@CS))
       then do(forward(X, exception(appoint-auditor), chief))
       else do(+role(auditor)), do(deliver).

Acceptance of the appointment to be the auditor.

R6. sent(X, delegate-supervisor(B),Y) :- role(supervisor)@CS, budget(B)@CS,
                                 do(-role(supervisor)), do(-budget(B)), do(forward).

A supervisor can delegate its supervisor’s role to others, along with the remaining budget.

R7. arrived(X, delegate-supervisor(B),X)
    :- if (role(auditor)@CS|not(type(management)@CS))
       then do(forward(X, exception(failed-delegation(B)), chief))
       else do(+role(supervisor)), do(+budget(B)),
              do(forward(Y, delegate-supervisor(X,Y,B), chief)), do(deliver).

Acceptance of the supervisor delegation, along with the budget.

R8. arrived(X,M,chief) :- (if (M=exception(failed-delegation(B))) then
do(-sAppointed)), do(deliver).

The arrival of the exception message at the chief.

Figure 3: Law PP
Note that the term role (supervisor) in the control state of an agent represents that it is current acting as the supervisor, and the term budget (B) indicate that it has B amount of purchasing budget, for distribution among the buyers.

Similarly, the chief can appoint any management type of agent x to be an auditor, if x doesn’t hold the role of supervisor, as in Rule R4 and R5. Note, the chief can appoint any number of auditors, which is different from the appointment of the supervisor. Also, to simplify our law, we didn’t specify the privileges of the agent with the role of auditor.

\[ \text{R9. } \text{sent}(X, \text{assign-budget}(B_1),Z) \leftarrow \text{role(supervisor)}@CS, \text{budget}(B)@CS, B=B_1, \right. \\
\left. \quad \text{do(decr(budget}(B),B_1)), \text{do(forward)}. \right. \]

Only a supervisor can assign the budget to the buyers, by taking it from its own budget.

\[ \text{R10. } \text{arrived}(X, \text{assign-budget}(B_1),Z) \leftarrow \text{if(not(type(staff)}@CS)) \right. \\
\left. \quad \text{then do(forward(Self,exception(assign-budget}(B_1)),chief)) \right. \\
\left. \quad \text{else if(budget}(B)@CS) \text{then do(incr(budget}(B),B_1)) \text{else} \right. \\
\left. \quad \text{do(+budget}(B_1)). \right. \]

The buyer accepts the assigned budget from the current supervisor.

\[ \text{R11. } \text{sent}(X, \text{purchase-order}(\text{specs}(S),\text{payment}(P)),V) \leftarrow \text{type(staff)}@CS, \right. \\
\left. \quad \text{budget}(B)@CS, B=P, \text{do(decr(budget}(B),P)), \text{do(release).} \right. \]

The buyer can issue the purchase order to the vendor if it has enough budget for the payment.

(c) Delegation, by transfer, of the supervisory role: By Rule R6 and R7, a supervisor can delegate its supervisory role to other management members by sending the delegate-supervisor(B) message, where B is its current purchasing budget. The very sending of such a message would remove the supervisory status, and the budget from the sender, thus observing Points 4a and 4b of policy PP. When this message arrives at its destination y, it will make y into the new supervisor, with the budget just removed from the sender—this, unless y is not a management agent, or if it holds the auditor role. In the latter case an exception message will be sent to the chief—and would be received by the chief by Rule R8—allowing the chief to make another appointment of a supervisor.

(d) Assignment of budgets and issuing POs: Only the supervisor can assign the purchasing budget to the buyers by sending message assign-budget(B1), under Rules R9 and R10. If the message receiver is not a staff, then an exception message will be forwarded to the chief, otherwise those assigned budget will be represented as the term budget(B) in the control state of the buyer.

Next, by Rule R11, a buyer can issue a purchase order, to any vendor, for a price that is no higher than its current budget, which is decremented appropriately. Note that although
the supervisor also has the budget in its control-state, it can’t use that budget to issue the purchase order because it doesn’t have the staff term.

Finally, we point out that according to Rule R11, the PO is released to the vendor, i.e., sent as an unregulated TCP/IP message, and not as an LGI-message. This is because we assumed that the vendors do not operate under LGI. Unfortunately this would not give the vendor any degree of confidence that the PO it got has been issued properly by a buyer of enterprise \(E\), nor would the vendor be able to prove this in a court of law. To provide the vendor with more trustworthy PO we will have the buyer send along with the PO an LGI-generated certificate, as discussed below.

4.2 The Regulated Creation of Delegation Certificates

Note that we have dispensed with the use of certificate within an LGI-community, carrying out delegation by changing the control-state of agents. But for a member of an LGI-community to communicate with unregulated agents, it may have to generate its own certificates. We will show now how this could help solving our problem with sending POs to the unregulated vendors.

\[ R11': \text{sent}(X, \text{purchase-order(specs}(S), \text{payment}(P)), V) \]
\[ :- \text{type(staff)}@CS, \text{budget}(B)@CS, B \geq P, \text{do}(\text{decr(budget}(B), P)), \]
\[ \text{POcert=certificate(issuer(ThisController), subject(Self), attributes([law(ThisLaw), purchaseOrder(specs(S), payment(P), vendor(V)), ...]), do(release(X, [purchase-order(specs(S), payment(P)), createCertificate(POcert), C-certificate]), V)).} \]

The buyer \(X\) can send the authorized purchase order to vendor \(V\) with two certificates: A certificate called \(C\)-certificate that authenticates the controller \(T_x\) that mediates interactions with agent \(x\); and a certificate signed by the controller, authenticating the PO—it is called here \(POcert\).

Figure 5: The issuing of purchase orders along with certificates

Let us replace Rule \(R11\) in Figure 4 with Rule \(R11'\) in Figure 5. With this new rule, every PO sent by a member \(x\) of the \(PP\)-community to an unregulated vendor \(v\), is accompanied by two certificates, as follows: (1) the certificate issued by a CA identified by its public key \(publicKey1\) (as defined in the law clause of law \(PP\) in Figure 3), which authenticates the controller \(T_x\) that mediates interactions with agent \(x\); and (2) a certificate issued by \(T_x\), containing the PO and the text of the law \(PP\)—which, effectively asserts that this PO has been issued legally under law \(PP\).

So, the vendor is getting here a chain of certificates, in the sense of CBD. And if the vendor trusts the CA that authenticates the controller, and if it trusts the logic of law \(PP\)—which it can examine, because it has been sent along with the PO\(^4\)—then it should be confident that the buyer \(x\) does have enough budget to cover the cost of the PO. This is because this law does not allow a buyer to pay for more than it has in its budget.

\(^4\)When the law needs to be private, only the hash of the law is being sent.
5 The Nature of Delegation under LGI, and its Complementarity with CBD

We are in a position now to reflect on the nature of distributed delegation under LGI, and on its complementary relationship with certificate-based delegation. We start with the former.

5.1 On the Nature of Regulated Delegation, under LGI

We will make here several observations about delegation under LGI, illustrating them via our case study.

(1) LGI does not use certificates to represent rights, or to delegate them. The rights of an agent are represented by its local control-state, and their semantics is defined by the law in question. For example, under law $PP$, the right of an agent to be a supervisor is represented by the term $\text{role(supervisor)}$ in its control-state. Delegations are, therefore, carried out by effecting an appropriate changes in the state of some agents. Thus, under law $PP$, for a supervisor to delegate its supervisory role to an agent $x$, it sends $x$ a message which, upon arrival, would add the term $\text{role(supervisor)}$ to its state.

(2) As a consequence of the above, there is no primitive notion of delegation in LGI mechanism per se, and none is required. There can be many circumstances which would allow one agent to confer some rights on another. Thus, one may have pure delegation, where one agent gives some of its own rights, or all of them, to another—as in the case of transferring the supervisor rule under $PP$. Or one may have a law that allows one agent to provide others with rights it does not have for itself. This is the case with the chief, under law $PP$, appointing somebody to the role of supervisor. These two types of “delegation,” and others, can represented in a unified manner, once they are defined by the law of the given community.

(3) An LGI law is communal, regulating the entire process of delegation (of various kinds) within the community governed by it. This provides LGI with the ability to support properties such as non-monotonicity, delegation by transfer, quotas over the number of times a right can be used, and others that are not supported by traditional CBD.

(4) The semantics of delegation is firmly defined by the law at hand, and can be relied upon by delegators and delegatees. For example, when the chief, under law $PP$, provides a budget to the supervisor it appoints, it can be confident that whatever is done with this budget, and among which buyers it would be distributed, it would be the upper limit for the cumulative cost of the purchase orders made by members of this community.

(5) The ability of an LGI law to regulate delegation often eliminates the need to carry around the history of delegation, all the way to its source, which under CBD is accomplished by chains of delegation certificates. Two examples, in the context of law $PP$, would illustrate this point.
First, consider a current supervisor \( s \), who may have obtained his baton at the end of a long sequence of transfers, starting with the original supervisor appointed by the chief. All \( s \) needs to keep in its control-state is the term \( \text{role(supervisor)} \), because there is no way for it to obtain this term, except via a legal sequence of transfers, mandated by this law.

Second, when sending a PO to the vendor, all one has to do is to send a pair of certificated (see Section 4.2) proving, in effect, that the PO is sent by a bona fide controller, interpreting law \( \mathcal{PP} \). This should convince the vendor that the buyer has enough funds in its budget to cover the cost of the PO, even though it has no information about the sequence of delegations of budgets, from the chief, all the way to the requesting buyer.

(6) Finally, we point out that despite our ability to regulate delegation (and other activities) under LGI, our access-control mechanism is quite open, flexible, and scalable for a wide range of policies. In particular we note that anybody can join the \( \mathcal{PP} \)-community, anytime, just by finding a controller certified by the CA mandated by law \( \mathcal{PP} \), and by presenting the required certificates signed by the CA called “admin.” The scalability of the mechanism is discussed in the Appendix, and in [9].

5.2 On the Complementarity of LGI and CBD

These two mechanisms are complementary in three different senses. First, their domain of applicability are complementary: LGI is intended to be used for communal policies, and for server-centric policies that require control over the process of delegation—neither of which is served well by CBD. CBD, on the other hand, could be the better choice for server-centric policies that do not require control over the delegation process; a good example of such a policy has been introduces in [3].

Second, LGI is itself based on CBD. Namely, each of the controllers, that collectively form the infrastructure of LGI, needs to be authenticated to be trusted. Our example law, \( \mathcal{PP} \), shows how such authentication is done via a single certificate. But a more sophisticated use of CBD is possible, and indeed, required when laws are organized into hierarchies (see [1]).

Third, LGI provides means for using delegation certificates, and for producing them, in a regulated manner, as follows:

- LGI employs CBD in order to provide distinctive characteristics to certain members of a given community. For example, law \( \mathcal{PP} \) makes a distinction between agents that belong to the management of the enterprise, and those that belong to its staff, on the basis of the certificate they provide. This is a very simple example of such use of certificates. Indeed, like Keynote [4], an LGI-law can require a whole chain of delegation-certificates in order to provide a given agents with certain privileges.

- In order to communicate with agents that employ CBD, but do not operate under LGI, an LGI agent can create delegation certificates, as has been illustrated in Section 4.2.
6 Conclusion

Certificate-based delegation (CBD) is a critical element in distributed access control, providing it with flexibility and scalability. But despite its elegance and effectiveness, CBD has inherent limitations that restrict its application domain. These limitations include, among others: lack of support for non-monotonic policies, such as separation of duties; the inability to support the transfer of privileges, where the delegator loses the privilege it delegates; and the lack of support for quotas, i.e., restrictions on the number of time a given privilege can be exercised.

This paper describes an approach to the distributed delegation problem, which shares much of the flexibility and scalability of CBD, but is not encumbered by limitations such as above. This approach is based on the distributed control mechanism called law-governed interaction (LGI), which employs a distributed set of reference monitors.

The resulting treatment of distributed delegation is not a replacement for CBD, but is complementary to it. First, because their range of suitable application domains is complementary. Second, because LGI itself is based on CBD, and uses it explicitly, when appropriate.
Appendix

A  On LGI Laws, and their Decentralized Enforcement

This appendix complements our brief introduction to LGI in Section 3, by elaborating on the structure of LGI laws, on the distributed enforcement of laws, on the deployment of the infrastructure of LGI, and on the treatment of certificates. For more information about this mechanism the reader is referred to [9, 2, 1].

A.1  On the Nature of LGI Laws, and their Decentralized Enforcement

The function of an LGI law $L$ is to regulate the exchange of $L$-messages between members of a community $C_L$. Such regulation may involve (a) restriction of the kind of messages that can be exchanged between various members of $C_L$, which is the traditional function of access-control policies; (b) transformation of certain messages, possibly rerouting them to different destinations; and (c) causing certain messages to be emitted spontaneously, under specified circumstances, for monitoring purposes, say, via a mechanism we call obligations.

A crucial feature of LGI is that its laws can be stateful. That is, a law $L$ can be sensitive to the dynamically changing state of the interaction among members of $C_L$. Where by “state” we mean some function of the history of this interaction, called the control-state (CS) of the community. The dependency of this control-state on the history of interaction is defined by the law $L$ itself. For example, under law $PP$ introduced in section 4, as a formalization of our example $PP$ policy, the term role(supervisor) in the control-state of an agent denotes that this agent has been appointed or delegated as the supervisor in the community.

But the most salient and unconventional aspects of LGI laws are their strictly local formulation, and the decentralized nature of their enforcement. To motivate these aspects of LGI we start with an outline of a centralized treatment of interaction-laws in distributed systems. Finding this treatment unscalable, we will show how it can be decentralized.

On a centralized enforcement of interaction laws: Suppose that the exchange of $L$-messages between the members of a given community $C_L$ is mediated by a reference monitor $T$, which is trusted by all of them. Let $T$ consist of the following three part: (a) the law $L$ of this community, written in a given language for writing laws; (b) a generic law enforcer $E$, built to interpret any well formed law written in the given law-language, and to carry out its rulings; and (c) the control-state ($CS$) of community $C_L$ (see Figure 6(a)).

The structure of the control-state, and its effect on the exchange of messages between members of $C_L$ are both determined by law $L$. For example, under law $PP$, a message...
delegate-supervisor will cause the specific term role(supervisor) be removed from the CS of the sender and added to that of the receiver if the delegation succeeds.

This straightforward mechanism provides for very expressive laws. The central reference monitor $T$ has access to the entire history of interaction within the community in question. And a law can be written to maintain any function of this history as the control-state of the community, which may have any desired effect on the interaction between community members. Unfortunately, this mechanism is inherently unscalable, as it can become a bottleneck, when serving a large community, and a dangerous single point of failure.

Moreover, when dealing with stateful policies, these drawbacks of centralization cannot be easily alleviated by replicating the reference monitor $T$, as it is done in the Tivoli system [7], for example. The problem, in a nutshell, is that if there are several replicas of $T$, then any change in $CS$, would have to be carried out synchronously at all the replicas. Such maintenance of consistency between replicas is very time consuming, and is quite unscalable with respect to the number of replicas of $T$.

Fortunately, as we shall see below, law enforcement can be genuinely decentralized, and carried out by a distributed set $\{T_x | x \in C\}$ of, what we call, controllers, one for each members of community $C$ (see Figure 6(b)). Unlike the central reference monitor $T$ above, which carries the CS of the entire community, controller $T_x$ carries only the local control-state $CS_x$ of $x$—where $CS_x$ is some function, defined by law $L$, of the history of communication between $x$ and the rest of the $L$-community. In other words, changes of $CS_x$ are strictly local, not having to be correlated with the control-states of other members of the $L$-community. However, such decentralization of enforcement requires the laws themselves to be local, in a sense to be defined next.

The local nature of LGI laws: An LGI law is defined over a certain types of events occurring at members of a community $C$ subject to it, mandating the effect that any such event
should have. Such a mandate is called the **ruling** of the law for the given event. The events subject to laws, called **regulated events**, include (among others): the **adoption** of the law by the agent, 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**. The operations that can be included in the ruling for a given regulated event, called **primitive operations**, are all local with respect to the agent in which the event occurred (called, the “home agent”). They include, operations on the control-state of the home agent and operations on messages, such as **forward**, **deliver** and **release**. To summarize, an LGI law must satisfy the following locality properties:

(a) a law can regulate explicitly only **local events** at individual agents; (b) the ruling for an event $e$ at agent $x$ can depend only on $e$ itself, and on the **local control-state** $\mathcal{CS}_x$; and (c) the ruling for an event that occurs at $x$ can mandate only **local operations** to be carried out at $x$.

**Decentralization of law-enforcement:** As has been pointed out, we replace the central reference monitor $\mathcal{T}$ with a distributed set $\{T_x \mid x \in \mathcal{C}\}$ of controllers, one for each members of community $\mathcal{C}$. Structurally, all these controllers are generic, with the same law-enforcer $\mathcal{E}$, and all must be trusted to interpret correctly any law they might operate under. When serving members of community $\mathcal{C}_e$, however, they all carry the same law $\mathcal{L}$. And each controller $T_x$ associated with an agent $x$ of this community carries only the **local control-state** $\mathcal{CS}_x$ of $x$ (see Figure 6(b)).

Due to the local nature of LGI laws, each controller $T_x$ can handle events that occur at its client $x$ strictly locally, with no explicit dependency on anything that might be happening with other members in the community. It should also be pointed out that controller $T_x$ handles the events at $x$ strictly sequentially, in the order of their occurrence, and atomically. This, and the locality of laws, greatly simplifies the structure of the controllers, making them easier to use as our trusted computing base (TCB).

Finally, we point out that the LGI model is silent on the placement of controllers **vis-a-vis** the agents they serve, and it allows for the sharing of a single controller by several agents. This provides us with welcome flexibilities, which can be used to minimize the overhead of LGI under various conditions.
**On the structure and formulation of laws:** Broadly 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.)

More formally, an LGI law \( L \) is a function (called the *ruling* function) of the following form:

\[
    r = L(e, cs), e \in E, cs \in CS, r \in R
\]  

(1)

where \( E \) is the set of regulated events, \( CS \) is the set of control states, and \( R \) is the set of all possible sequences of operations that constitute the ruling of the law. Figure 7 illustrates the way that this function is used when it is deployed within a controller. When an event \( e \) occurs in this controller, it is supplied as an input to the law (function), along with the current control state \( cs \) maintained by the controller. The law then evaluated the ruling \( r \), and returns it to the controller, which carries it out.

Concretely, such a function can be expressed in many languages. Our middleware currently provides two languages for writing laws: Java, and a somewhat restricted version of Prolog. We employ prolog in this paper. In this case, the law is defined by means of a Prolog-like program \( L \) 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 \( cs \), producing the list of primitive-operations \( r \) 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 8.

**On the basis for trust between members of a community:** For a member of an \( L \)-community to trust its interlocutors to observe the same law, one needs the following assurances: (a) that the exchange of \( L \)-messages is mediated by correctly implemented controllers; (b) that these controllers are interpreting the *same law* \( L \); and (c) that \( L \)-messages are securely transmitted over the network. If these conditions are satisfied, then it follows that if \( x \) receives an \( L \)-message from some \( y \), this message must have been sent as an \( L \)-message; in other words, that \( L \)-messages cannot be forged.
Operations on the control-state

\[ t \in CS \] 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;

\[ t1 \leftarrow t2 \] replaces term \( t1 \) from the control state with term \( t2 \);

\[ \text{incr}(f(v1), v2) \] increases the argument \( v1 \) of functor \( f \) by \( v2 \);

\[ \text{decr}(f(v1), v2) \] decreases the argument \( v1 \) of functor \( f \) by \( v2 \);

Operations on messages

\[ \text{forward}(x, m, y) \] sends message \( m \) from \( x \) to \( y \); triggers at \( y \) an arrived\((x, m, y)\) event;

\[ \text{deliver}(x, m, y) \] delivers the message \( m \) from \( x \) to \( y \);

\[ \text{release}(x, m, y) \] sends message \( m \) from \( x \) to \( y \), in which \( y \) is an non-LGI agent;

Miscellaneous

\[ t \in L \] returns true if term \( t \) is present in list \( L \), and fails otherwise

\[ \text{imposeObligation}(oType, dt) \] causes the triggering of an obligationDue\((oType)\) event after time interval \( dt \).

\[ \text{repealObligation}(type) \] repeals all \( type \) obligations

Figure 8: Some primitive operations in LGI.

Broadly speaking, these assurances are provided as follows: Controllers used for mediating the exchange of \( L \)-messages authenticate themselves to each other via certificates signed by a certification authority specified by the value of the \texttt{ca}\ attribute in the \texttt{law} clause of law \( L \) (see, for example, Figure 3, in the case of law \( PP \)). Note that different laws may, thus, require different certification levels for the controllers used for its enforcement. Messages sent across the network are digitally signed by the sending controller, and the signature is verified by the receiving controller. To ensure that a message forwarded by a controller \( T_x \) under law \( L \) would be handled by another controller \( T_y \) operating under the same law, \( T_x \) appends a one-way hash \cite{10} \( H \) of law \( L \) to the message it forwards to \( T_y \). \( T_y \) would accept this as a valid \( L \)-message under \( L \) if and only if \( H \) is identical to the hash of its own law.

**The deployment of LGI** All one needs for the deployment of LGI is the availability of a set of trustworthy controllers, and a way for a prospective client to locate an available controller. This can be accomplished via one or more controller-services, each of which maintains a set of controllers, and one or more certification authorities that certifies the correctness of controllers. For an agent \( x \) to engage in LGI communication under a law \( L \), it needs to locate a controller, via a controller-service, and supply this controller with the law \( L \) it wants to employ and some possible certificates that required by \( L \) for the law adoption. E.g., in law \( PP \), an agent \( x \) can adopt the law and so join the \( C_{PP} \) community only if it can present the certificate issued by the enterprise \( E \)’s \texttt{admin} certifying that \( x \) is either a \texttt{management} or a \texttt{staff} of \( E \).
A.2 Acceptance and Creation of Certificates under LGI

Under LGI, the certificate can be used to control the adoption of the law, i.e., serves as the community admission control, as we discuss above. An agent can also submit various certificates after it joins the community, allowing the law to make distinctions between agents.

The submission by an agent \( x \), operating under law \( L \), of a certificate \( \text{Cert} \) to its controller, has the following effect: An attempt is made to confirm that \( \text{Cert} \) is a valid certificate, duly signed by an authority that is acceptable to law \( L \), i.e., an authority that is represented by one of the authority-classes in the preamble to the law (See Figure 3 for an example). If this attempt is successful\(^5\), then the certificate will be translated into the following certificate format message:

\[
\text{certificate}(\text{issuer}(I), \text{subject}(S), \text{attributes}(A)).
\]

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 \( \text{attribute}(\text{value}) \) terms. For example, the attributes of a certificate might be the list \([\text{type}(\text{staff})]\), asserting that the type of the subject in question in this community is a staff. 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.

What happens after the certificate is checked to be valid depends, of course, on the law. In the case of law \( PP \) of Figure 3, for example, the following would happen when an agent adopts the law, by presenting its certificate issued by the \( \text{admin} \) to certify its type—either as \( \text{management} \) or \( \text{staff} \): the term \( \text{type}(\text{management}) \) or \( \text{type}(\text{staff}) \) is set in the control-state of the agent in question. Due to lack of space, we don’t discuss here how to deal with the expiration and revocation of the accepted certificate in LGI. For those issues, the reader is referred to [2].

A LGI law can not only accept the certificates, but also create the certificates. Such creation of certificates is also governed by the law, which can give the formal semantics of the created certificates. E.g., in the law \( PP \), when a buyer \( x \) is authorized to issue a purchasing order, a purchase order certificate can be created and issued by \( x \)’s controller \( T_x \) interpreting law \( PP \). This is done by calling a built-in function \( \text{createCertificate} \) with a certificate format argument \( \text{POcert} \), where \( \text{POcert} \) has the public key of \( T_x \) as the issuer, the public key of \( x \) as the subject and its attributes include the specification and payment of that PO. This capability allows agents operating under LGI to interact smoothly with agents that do not.

\(^5\)If the the certificate is found invalid then an \textit{exception-event} is triggered.
References


