

Formal Support for Certificate Management Policies

Victoria Ungureanu
Department of MSIS
Rutgers University
180 University Ave.
Newark, NJ 07102, USA
ungurean@rbs.rutgers.edu

Abstract

Traditionally, creation and revocation of certificates are governed by policies that are carried manually, off-line, by trusted agents. This approach to certificate management is appropriate for many current applications, where these policies cannot be verified automatically (e.g. require verification of non-digital credentials). But it is expensive, time consuming and error-prone for the growing class of applications where certificate management policies can be formalized and carried-out automatically. We argue that, in these cases, creation and revocation of certificates could be viewed as any other on-line service available in a system. Access to these particular service instances could be regulated much in the same manner as file access or resource allocation.

This paper proposes a formulation for certification and revocation policies, and a framework for their support. In this framework, certificate management policies are enforced by generic policy engines, wrapped around certification authorities and revocation servers. The proposed framework is easy to deploy, requiring no modifications of current public key infrastructure (PKI). Moreover, we show that this framework is quite affordable, even in its present, experimental stage.

Keywords: certification, revocation, PKI, certificate management policy, policy formulation, policy enforcement.

1 Introduction

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

To date, several certificate frameworks [6, 16, 26] and revocation mechanisms [11, 23, 24] have been proposed and extensively studied. What interests us here is an orthogonal aspect of certificate management, which has received considerably less attention, namely the formulation and enforcement of policies governing certification and revocation.

Certification. A certificate is signed on behalf of an organization by a *certification authority* (CA) if a number of provisions are met. Typically these provisions, which are specified by a Policy Certification Authority (PCA), are verified manually, off-line, by trusted agents. A CA serves a request to create a certificate only if the request is made by an agent trusted to carry out the terms of the PCA [20, 22].

Such an approach is indeed necessary if a PCA cannot be verified automatically—for example, if a PCA requires non-digital credentials, like birth certificates, driver’s licenses, or academic diplomas. However, in the many cases when the PCA terms can be verified automatically, such enforcement of policies is time consuming, expensive and error-prone.

The drawbacks of an entirely manual approach towards certification can be illustrated by the following example. In business-to-business (B2B) e-commerce supplier-enterprises often require that purchase orders from their clients are signed by CAs established by the respective client-enterprises [18]. From the supplier point of view, this requirement has several important benefits, including: (a) it ensures that a purchase order (PO) is valid, and (b) it makes PO validation simple, in that only one signature needs to be verified. From the client-enterprise point of view, this requirement calls for the signing of every PO. But if every little purchase is manually verified by a designated authority, the daily routine of a company, which makes a large number of purchases, is bottlenecked.

In practice, there are cases when compliance with PCAs could be verified automatically. For example, consider a large, geographically distributed company, called ACME, which is required by some of its suppliers to certify POs. Suppose further that ACME’s PCA regarding PO certification states that only duly appointed purchase-officers may issue POs. If the status of purchase-officers is established by digital credentials, this PCA can be verified automatically, thus making purchasing considerably more efficient.

Revocation. Certificates might become invalid for various reasons and should be revoked. For example, in the scenario presented above, a purchase-officer might lose or otherwise compromise his private key, or he might leave the company. Most revocation mechanisms rely on the existence of *certificate revocation lists* (CRLs) maintained by trusted revocation servers. In these mechanisms, revocation is carried out as follows: the certificate owner, or

another designated authority, sends the server a signed message identifying the certificate to be revoked [1, 20, 25]. Upon receipt of the message, the revocation server updates its CRL and disseminates the information.

But, relying on the certificate owner for revocation is problematic because the owner can be entrusted to report an invalid certificate only in a limited number of cases. For example, in the ACME scenario, one cannot rely that a purchase-officer revokes his certificate when he leaves the company. And requiring that each certificate be revoked by a designated agent, usually a person, is an expensive, error-prone process. An alternative to revocation is issuing certificates with short validity periods [21]. But this solution requires frequent renewal of certificates, and thus is computationally expensive if the number of certificates issued on behalf of an organization is large [25]. Moreover, there are cases, which require immediate revocation, and where using short validity certificates might lead to undesirable effects.

As an example where having short validity periods cannot be used as substitute for revocation, consider a distributed database containing data regarding various commercial companies. Assume that access to this database is subject to the well-known *Chinese Wall* policy [4]. Under this policy, companies are partitioned into a set of disjoint “competition groups”, where each group contains companies that compete with each other in the market place. The Chinese Wall requires that once an agent gets information about a company in a group G , he should not be allowed to get information about any other company in G . If agent privileges are established by certificates with limited time periods, then a malicious agent may improperly gain access to data of several companies in a competition group G . This is because, during a certificate validity period, an agent may present the certificate to different servers of the database, maintaining data of various companies in G . Since the certificate is valid, several servers might authorize access.

This example shows that having short-lived certificates as substitute for revocation may not be appropriate. And we have argued that it may be unsafe to rely on the owner (or a designated authority) to send revocation notices. These problems could be alleviated by allowing other agents to report that the status of a certificate has changed and it should be revoked. For example, in the Chinese Wall scenario, a database server could request the revocation of a group certificate, at the time the server grants access to a company data based on this certificate. This would make revocation more reliable because servers, unlike owners, do not have a vested interest in certificates, and thus are more likely to report the change in a certificate status.

Formal support for certificate management policies. We argue that creation and revocation of certificates could be viewed as any other on-line service available in a system; access to these particular service instances could be regulated much in the same manner as file access or resource allocation. Namely, this can be achieved by formally expressing certificate management policies and by providing a generic mechanism for enforcing them.

Like traditional access control policies, certificate management regulations should specify who is authorized to request what type of service. Examples of such regulations, drawn

from the ACME scenario, include: a request to sign a PO is authorized if issued by a purchase-officer; and a request to revoke a purchase-officer’s certificate is authorized if made by a member in the personnel department, who is presumably aware of the status of the purchase-officer. Furthermore, current certification and revocation regulations may be implemented as specific policies. For example, an enterprise can formalize a policy stating that the system administrator is allowed to create any type of certificate, or that an owner is allowed to revoke his certificates.

This paper describes a format for certificate management policies (CMPs) and a framework for their support. In this framework, CMPs are enforced by generic policy engines, called CMP-engines. Access to the services provided by CAs or revocation servers, referred henceforth as public-key (PK) servers, is regulated by wrapping CMP-engines around them. A request addressed to a PK-server is thus first intercepted by the corresponding CMP-engine; the CMP-engine delivers the request to the intended destination only if it is sanctioned by the CMP established in it.

The proposed framework can be used not only to control access to services provided by PK-servers, but also to extend, to a certain degree, their functionality. Applications may require services that are not supported by the existing PK-servers. In this case, an alternative to building a new server, is to implement into a CMP the desired functionality, and wrap the PK-servers with policy engines enforcing the CMP in question.

To illustrate the need for such extensions consider the interface provided by revocation servers. This interface generally allows to test the validity of a certificate, and to revoke a certificate, but does not support means for synchronizing these requests. But synchronization of test and revoke requests might be required to prevent race conditions in distributed applications. A case to point is the Chinese-Wall policy described above. Its correct implementation requires that the testing and the revocation of a given group certificates are done in one atomic step. Short of that, it is possible that *several* servers will perceive a group certificate as valid, and consequently *several* requests will be incorrectly authorized.

The rest of the paper is organized as follows. Section 2 describes the manner in which certified management policies are enforced. Section 3 presents the language used to formulate CMPs; their formulation is illustrated in Section 4 by concrete examples. Section 5 shows how the functionality of PK-servers can be extended in the proposed framework, and introduces a CMP that synchronizes test and revoke operations. Section 6 presents the implementation of the mechanism and shows experimentally that the proposed CMP formulation lends itself to efficient enforcement. Section 7 discusses related work, and Section 8 concludes the paper.

2 The CMP Enforcement Mechanism

The enforcement mechanism relies on the existence of generic CMP-engines, wrapped around PK-servers, as illustrated in Figure 1. Such policy engines are trusted to carry out the terms of certified management policies established in them.

The placement of CMP-engines between the communication medium and PK-servers en-

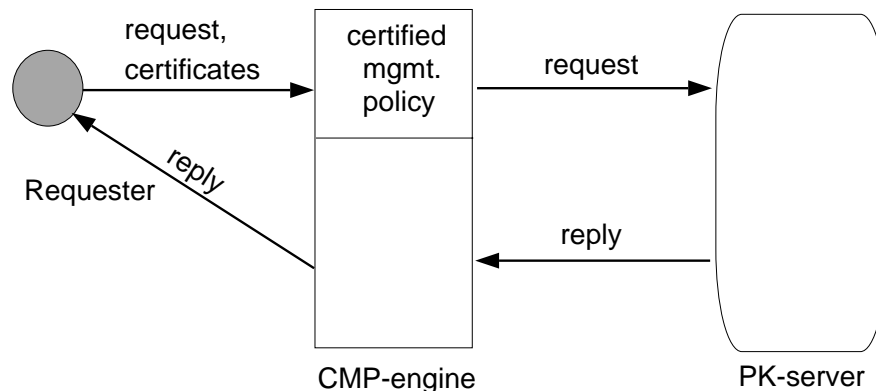


Figure 1: System architecture.

sures that any such server receives only requests sanctioned by the certified management policy carried out by the wrapper. More specifically, consider that a PK-server PKS (which can be either a CA or a revocation server) is wrapped by a CMP-engine E , enforcing a certificate management policy P . Assume further that a user U , sends to PKS a request R , which is accompanied by a list (possibly empty) of subject-certificates. Then the following steps occur:

1. Request R is intercepted by CMP-engine E .
2. For each subject certificate SC accompanying the request, E attempts to confirm that SC is valid. This implies verifying that the signature is correct, SC belongs to U , and SC has not been revoked.
3. E evaluates the rules of the certified management policy P . If the request is sanctioned by P , it is sent to PKS . Otherwise the request is discarded, and U is notified of the outcome.
4. PKS serves the request. The reply for this request is also captured by E , and may be also evaluated with respect to P . Finally, U is notified of the outcome.

An important difference between a CMP-engine and other enforcement mechanisms proposed recently (e.g. [2, 3, 5, 7, 12, 14, 15]) is that the latter type *calls only for evaluating the request*. In contrast, under the framework presented here a request may trigger two policy evaluations: one when the request is intercepted and another one when the corresponding reply is made. This approach considerably extends the range of supported policies. In particular, evaluating the reply is instrumental in implementing synchronization operations. However, if a policy calls only for the evaluation of the request, a CMP-engine can be configured to deliver the reply without evaluating it, thus yielding a better overall performance.

3 Formulation of Certificate Management Policies

Certificate management policies can be quite naturally expressed by means of any formal language supporting *event-condition-action* (ECA) kind of rules. We are using here an extension of a language devised for support of control policies [19] built on top of Prolog. In this language, the treatment of a request is formalized by Prolog-predicates of the form:

```
request([Requester,LCert],R,PKS):-  
    condition-1,...,condition-k,  
    provision-1,...,provision-n.
```

In this rule, `Requester` denotes the agent making request `R`; `LCert` is the list of the valid certificates presented by `Requester`; and `PKS` denotes the PK-server wrapped by the policy engine enforcing the CMP in question. This rule stipulates that if `condition-1` through `condition-k` are satisfied then `provision-1` through `provision-k` are carried out. Conditions used in the body of a rule may refer to the request and the certificates presented. Provisions are additional actions stipulated by the policy, and may include such actions as delivering a sanctioned request to the associated PK-server, or delivering the server's reply to the agent that made the request.

If a policy calls also for reply evaluation, the treatment of a reply is formalized by Prolog-predicates of the form:

```
reply(PKS,Reply,Requester):-  
    condition-1,...,condition-k,  
    provision-1,...,provision-n.
```

In order to support conditions and provisions, the language proposed here defines a set of additional predicates, called primitive operations. Some of the primitive operations currently supported are displayed in Figure 2, and are briefly described below:

- **Operations on certificates.** These operations probe the content of a certificate and determine the issuer, or the values of the attributes attested by it. The operations on certificates include: `issuer(PK,Cert)` which binds variable `PK` to the public key of the issuer of certificate `Cert`, and `bind(Attr,Val,Cert)`, which binds `Val` to the value of attribute `Attr`, in certificate `Cert`. Since some attributes are used frequently, the language provides the following specializations of `bind`: `name(N,Cert)`, which binds variable `N` to the value of attribute `name` in certificate `Cert`, and `role(R,Cert)`, which binds `R` to the value of attribute `role` in certificate `Cert`. (We mention that certificates are translated into an internal format, essentially a Prolog list, before being passed to the Prolog interpreter. The predicates defined above are implemented as operations on lists.)

For example, assume that variable `Cert` is bound to a certificate signed with public key `rsa:3:ab1cd2ef` (real keys are, of course, much longer). Suppose further that this certificate contains the following statement: `name(johnDoe), role(purchaseOfficer), employer(acme)`. Then, after carrying out operation `issuer(PK,Cert)`, the value of `PK`

Operations on certificates	
<code>issuer(PK,Cert)</code>	binds <code>PK</code> to the public key of the issuer of certificate <code>Cert</code> ;
<code>bind(Attr,Val,Cert)</code>	binds <code>Val</code> to the value of attribute <code>Attr</code> in <code>Cert</code> , if it exists; fails otherwise;
<code>name(N,Cert)</code>	binds <code>N</code> to the value of attribute <code>name</code> in <code>Cert</code> , if it exists; fails otherwise;
<code>role(R,Cert)</code>	binds <code>R</code> to the value of attribute <code>role</code> in <code>Cert</code> , if it exists; fails otherwise;
Miscellaneous operations	
<code>deliver(S,M,D)</code>	delivers message <code>M</code> , ostensibly sent by <code>S</code> , to destination <code>D</code> ;
<code>T@L</code>	returns true if term <code>T</code> is present in list <code>L</code> , and fails otherwise.

Figure 2: Some primitive operations.

is `rsa:3:ab1cd2ef`. Similarly, the effect of operations `name(N,Cert)`, `role(R,Cert)` and `attr(employer,V,Cert)` is to bind variable `N` to `johnDoe`, variable `R` to `purchaseOfficer`, and variable `V` to `acme`.

- **Miscellaneous operations.** Other supported operations include: (1) `deliver(S,M,D)`, which delivers message `M`, sent by `S`, to destination `D`, and (2) `T@L`, which succeeds if term `T` is present in list `L` and fails otherwise.

The use of these primitives will be illustrated by presenting several concrete examples of CMPs in Sections 4 and 5.

4 Establishing Certificate Management Policies

We show here how certificate management regulations can be established by presenting informally a policy and by showing how it can be implemented as CMPs.

4.1 ACME’s Certificate Management Policy

To illustrate certificate management regulations we present a thorough description of the certification and revocation rules of ACME, introduced schematically in Section 1. This policy assumes that ACME’s public-key infrastructure consists of the following servers: (a) two certification authorities, whose public keys are denoted by the shorthand notation `pk1_acme` and `pk2_acme`, and (b) two revocation servers, having addresses `rev1.acme.com` and `rev2.acme.com`. (ACME’s PKI could consist in any number of CAs and revocation servers. We have chosen only two for simplicity.) Given this infrastructure, the certificate management policy of ACME can be stated as follows:

1. *The identities and roles of the various ACME employees must be validated via digital certificates issued by any of ACME’s certification authorities. Specifically, the status of a purchase-officer is established by a digital certificate containing the attribute `role(purchaseOfficer)`, and the status of a member in the personnel department is established via a certificate containing the attribute `department(personnel)`.*
2. *Only Jane Doe, the head of the personnel department, is authorized to create or to revoke certificates for personnel staff. An agent duly certified as a member in the personnel department is authorized to create or to revoke certificates for purchase-officers.*
3. *An agent duly certified as a purchase-officer is authorized to request the certification of a purchase order (PO). Additionally, certified POs should be sent, for monitoring purposes, to a designated agent called `auditor`.*

We will present in Sections 4.3 and 4.4 two CMPs, which implement the certification and revocation aspects of the policy above. To lay the ground for these examples, we start in Section 4.2 with a preliminary presentation of the interfaces provided by a public-key infrastructure.

4.2 Public-Key Infrastructure

Traditional public-key infrastructures (PKIs) assume the existence of the following entities: (1) certification authorities (CAs) which are entitled to sign certificates, and (2) revocation servers, which maintain and disseminate information related to certificate revocation. We proceed by describing the types of requests recognized by these servers.

Certification authority (CA). A CA responds to requests to sign arbitrary statements. The formalizations of the certificate management policies presented in this paper assume that this type of request is denoted by requests of the form

`certify(Stmnt)`,

where `Stmnt` is the statement to be certified. A CA responds to a `certify` request, with a message `certified(Stmnt,Cert)`, where `Cert` is a digital certificate for `Stmnt`, signed with the private key of the CA.

Revocation server. As we have mentioned above, a revocation server is designed to maintain information regarding certificate status. We assume that a revocation server recognizes the following requests:

- `test(Cert)`—a request to test the status of `Cert`. The revocation server is expected to reply with a message of the form `status(S,Cert)`, where `S` denotes the status of `Cert` (`S` can be either `valid` or `revoked`).

- `revoke(Cert)`—a request to revoke certificate `Cert`. The server is expected to respond to a `revoke` message with a message `revoked(Cert)`.

We point out that a PKI does not need to support the exact format for requests described above. One of the benefits of the framework presented here is that CMPs can be designed to use arbitrary format for PKI requests. Consequently, off-the-shelf CAs/revocation servers may be used without any modifications.

4.3 The CMP Regulating Certification

The CMP regulating certification, displayed in Figure 3, is enforced by the CMP-engines wrapping ACME’s certification authorities. This CMP has four rules, which are followed by explanatory comments (in italics). The discussion of the rules is organized as follows. We start by showing how requests to `certify` various types of statements are treated. We will then show how the CA responses to these requests are handled.

Rule $\mathcal{R}1$ regulates requests to sign certificates for personnel staff, i.e. requests to certify statements containing the term `department(personnel)`. The rule mandates that such a request is delivered to the intended CA only if the requester presents a certificate issued by `pk1_acme` or `pk2_acme`, which certifies the bearer to be Jane Doe. Rule $\mathcal{R}2$ regulates requests to sign purchase-officer certificates, i.e. requests to certify statements containing the term `role(purchaseOfficer)`. This type of request is delivered to the intended CA only if the requester presents a certificate issued by `pk1_acme` or `pk2_acme`, which certifies the bearer to be a member in the personnel department. Rule $\mathcal{R}3$ handles requests to certify a purchase order (PO), assumed to be of the form `po(Terms)`. This rule stipulates that a request to sign a PO is delivered to the CA only if the requester is authenticated as a purchase-officer by a certificate issued by `pk1_acme` or `pk2_acme`.

Finally, Rule $\mathcal{R}4$ deals with `certified` messages, sent by a CA in response to `certify` requests. This rule mandates that the reply should be delivered to the agent that made the request. Additionally, if the signed statement is a PO, then the certified PO is also delivered to `auditor`, thus realizing Point 3 of ACME’s policy.

It is worth pointing out that this simple policy is very difficult to support using currently available CAs. As we have already argued, certification policies are usually not formalized, but are manually verified by agents. If the PCA verification is delegated to few trusted agents, the activity of a large company, issuing many POs daily, may be bottlenecked. And delegating this task to many employees may be unsafe. In this example, allowing members in the personnel department and purchase-officers to create any type of certificates might lead to the ugly scenario where personnel staff issue purchase orders, and purchase-officers create employee certificates.

4.4 The CMP Regulating Revocation

The CMP-engines wrapping ACME’s revocation servers are enforcing the policy displayed in Figure 4. This CMP has three rules, implementing the revocation regulations stated

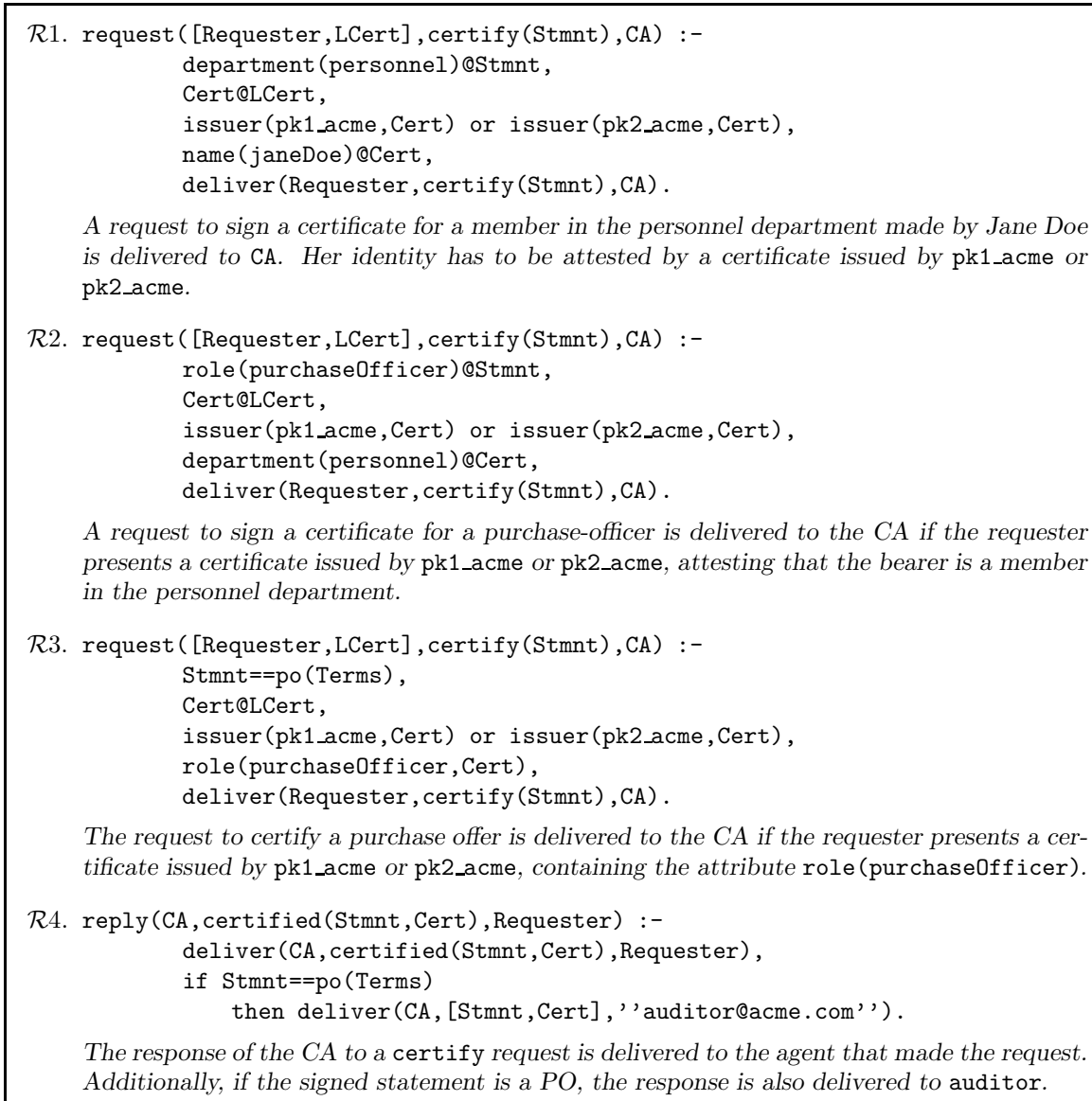


Figure 3: The CMP regulating certification.

informally in Section 4.1. Rule $\mathcal{R}1$ deals with requests to revoke a certificate for a member in the personnel department. These requests are delivered to the intended revocation server RS , only if the requester is properly authenticated to be Jane Doe. Requests to revoke a purchase-officer certificate are regulated by Rule $\mathcal{R}2$. By this rule, this type of requests is delivered to a revocation server RS , only if the requester is a member in the personnel department (i.e. he presents a certificate issued by `pk1_acme` or `pk2_acme`, which contains attribute `department(personnel)`). Finally, Rule $\mathcal{R}3$ stipulates that requests to test the

status of a certificate are delivered to the revocation server without further ado.

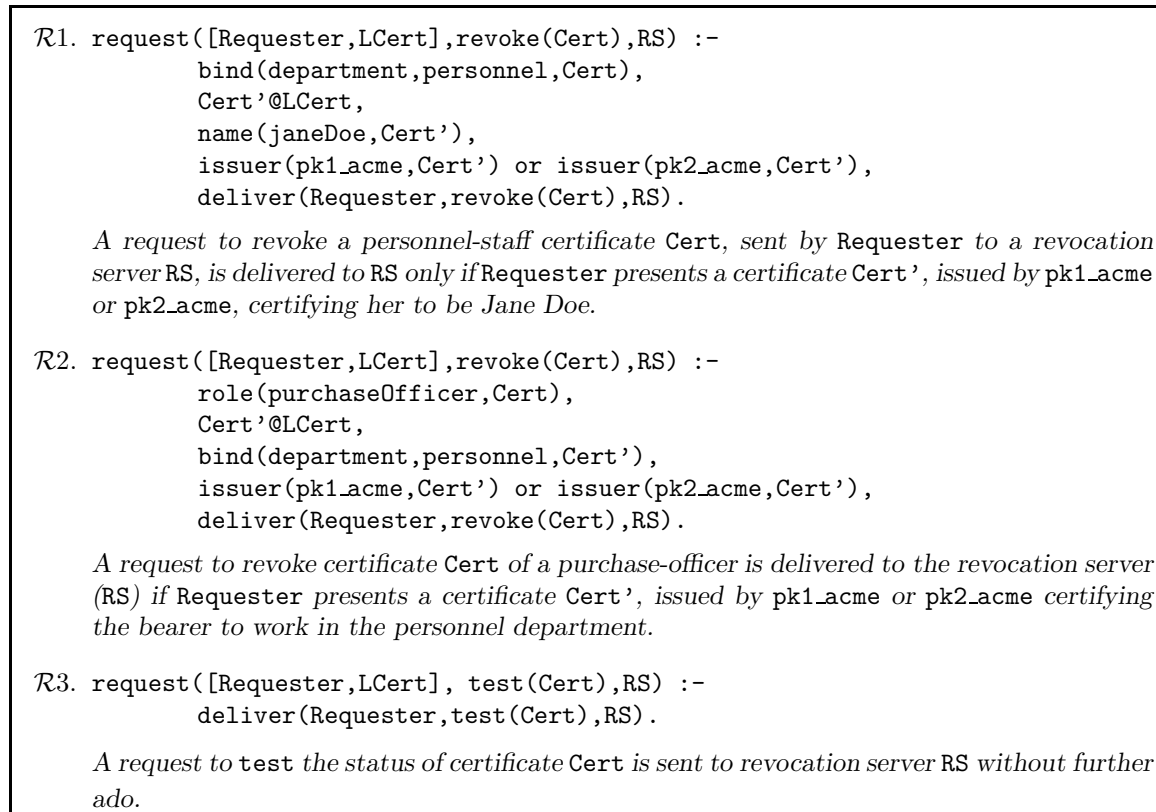


Figure 4: The CMP regulating revocation.

We end this discussion by emphasizing that this CMP does not provide any `reply` rules. Consequently, the CMP-engines enforcing it are configured to deliver the replies of revocation servers directly to requesters. In contrast, the CMP regulating certification provided both request and reply rules. This is necessary because the policy stipulates that certain actions are taken when a certificate is issued. Specifically, it required that certified POs are also delivered to `auditor`. This requirement can be fulfilled only if replies are also evaluated by CMP-engines.

5 Extending PKI Functionality: Means for Synchronization

It has been pointed out in Section 1 that applications may require services that are not supported by the existing PKI. A specific example of such a service is means for synchronization. Synchronization of test and revoke requests are needed, in particular, for the correct implementation of the Chinese Wall policy. We have argued that in this case, an alternative to building a new PK-server—an expensive and error-prone undertaking—is to

implement into a CMP the desired functionality, and wrap the PK-servers with CMP-engines enforcing the CMP in question.

In this section we show how a synchronization operation, which we call `test&revoke`, may be supported in this manner. This operation is designed to ensure that access to a shared resource (in this case the certificate to be tested and revoked) is done in a mutual exclusive manner. Specifically, a request to `test&revoke` a certificate `Cert` performs the following operations in *one atomic step*: (1) test if `Cert` is valid, (2) report the status of `Cert`, and (3) revoke `Cert`, if it is valid.

Assuming that a revocation server has the functionality described in Section 4.2, a CMP could implement a request to `test&revoke` a certificate `Cert` essentially as follows:

1. If the revocation server is not processing another operation on `Cert`, then deliver a `test(Cert)` request to it.
2. When the revocation server responds to the `test(Cert)` request, deliver the response to the requester. Furthermore, if `Cert` is valid, deliver a `revoke(Cert)` request to the revocation server.
3. If any other requests regarding `Cert` are intercepted while a `test&revoke(Cert)` operation is processed, block their processing for the time the revocation server uses `Cert`.

We draw the reader’s attention to the following points. First, this implementation achieves its purpose because the execution of a `test&revoke(Cert)` does not interleave with the execution of any other operation regarding `Cert`. Consequently, if several agents issue quasi-simultaneously `test&revoke(Cert)` requests, at most one agent perceives `Cert` as valid.

Secondly, from this brief description, it becomes apparent that a CMP implementing the `test&revoke` operation needs to record blocked requests and to know what operations are currently processed by the revocation server. In order to support the implementation of `test&revoke`, we are extending the formalism so that the rules of a CMP may take into consideration additional information—referred here as *control state*. A CMP whose rules use control state is called stateful. Stateful CMPs are described in Section 5.1, and are illustrated by presenting in Section 5.2 a CMP that implements the `test&revoke` operation.

5.1 Stateful CMPs

The control state of a CMP can represent such things as the operations a PK-server is currently processing, or the past actions of various agents. The semantics of control state for a given CMP is defined by its rules. For example, under the CMP implementing `test&revoke` operation, to be introduced in Section 5.2, the term `processing(Op,Cert)` in the control state denotes that the revocation server is performing operation `Op` on certificate `Cert`.

The control state of a CMP can change dynamically subject to the provisions of the CMP in question. Several additional primitive operations have been introduced to support updates

<code>addCS(T)</code>	adds term <code>T</code> to the control state;
<code>delCS(T)</code>	removes term <code>T</code> from the control state;
<code>T@CS</code>	returns true if term <code>T</code> is present in the control state and fails otherwise

Figure 5: Operations on control state.

of the control state, and to probe its content. These operations, displayed in Figure 5, include: (1) `addCS(T)`, which adds term `T` to the control state, (2) `delCS(T)`, which deletes term `T` from the control state, and (3) `T@CS`, which succeeds if term `T` is present in the control state (and fails otherwise).

5.2 The Implementation of the `test&revoke` Primitive

The previously defined `test&revoke` operation is implemented by the `CMP` presented in Figure 6. The rules of this `CMP` provide treatment only for `test&revoke` and `test` requests; `revoke` requests are not considered because their functionality is subsumed by `test&revoke`.

The control state of this `CMP` may contain terms of the form `processing(Op,Cert)` and `blocked(Requester,R,Cert)`. As it was already mentioned, the presence in the control state of the `processing(Op,Cert)` term denotes that operation `Op` is being performed on certificate `Cert`; we shall see that the presence of the term `blocked(Requester,R,Cert)` signifies that a request `R` made by `Requester` and pertaining to certificate `Cert` has been blocked. The following discussion of the rules of this `CMP` shows how such terms are added and removed from the control state, and how their presence affects the treatment of requests.

When a `Requester` makes a valid request `R` regarding certificate `Cert`, the `CMP` checks if any other operation `Op'` on `Cert` is currently being processed (Rule $\mathcal{R}1$). This checking is done by testing whether the control state contains the term `processing(Op',Cert)`. If the term is found, `R` is not delivered to the revocation server. Instead the information regarding this request is stored in the control state, by adding to it the term `blocked(Requester,R,Cert)`. However, if the control state does not contain a term `processing(Op',Cert)` then this rule mandates that `R` should be processed immediately.

We describe now how various requests are processed by this `CMP`. We start by considering that `R` denotes a `test&revoke(Cert)` request. Then, by Rule $\mathcal{R}2$, the following actions are taken: (1) a request to test the status of `Cert` is delivered to the revocation server, and (2) a term `processing(test&revoke,Cert)` is added to the control state, to record that a `test&revoke(Cert)` request is being processed. The presence in the control state of this term effectively prevents the `CMP`-engine to deliver any other requests regarding `Cert` to the revocation server (see Rule $\mathcal{R}1$ presented above).

When the revocation server responds to the `test(Cert)` request, the response is delivered to the intended destination (Rule $\mathcal{R}4$). Furthermore, if `Cert` is valid then a `revoke(Cert)` request is delivered to the revocation server, to complete the `test&revoke` operation. (If `Cert` is invalid, there is no need to revoke it.)

Rule $\mathcal{R}5$ handles the reply to the `revoke(Cert)` request. The arrival of this response signals the completion of the `test&revoke(Cert)` operation. Consequently, the rule calls for resuming a previously blocked request regarding `Cert`, if any. (The same action is called for when `Cert` is invalid.) By Rule $\mathcal{R}6$, this is done as follows. The CMP checks if there is a blocked request `R` regarding `Cert`, by testing whether the control state contains a term `blocked(Requester,R,Cert)`. If such a term is found, then the term `blocked(Requester,R,Cert)` is removed from the control state and the request is finally processed (see Rules $\mathcal{R}2$ and $\mathcal{R}3$). Additionally, by the same rule the term `processing(test&revoke,Cert)` is deleted from the control state, marking that the `test&revoke(Cert)` operation has been completed.

We describe now how this CMP handles the processing of `test` requests. Recall that processing of a `test(Cert)` is triggered if such a request is intercepted when the revocation server is not executing another operation on `Cert` (see discussion of Rule $\mathcal{R}1$ above) or when this request has been selected for processing after an operation on `Cert` has finished (see Rule $\mathcal{R}6$). Processing a `test(Cert)` request is handled by Rule $\mathcal{R}3$ as follows: (1) a term `processing(test,Cert)` is added to the control state, and (2) a request to test the status of `Cert` is delivered to the revocation server. When the `test(Cert)` request is completed (i.e. the revocation server sends a `status(S,Cert)` response) another request regarding `Cert` may be resumed (Rule $\mathcal{R}4$).

We conclude this discussion by pointing out that this CMP is incomplete in that it does not specify the conditions under which various requests are authorized. An application that uses this CMP needs to formalize the `authorize` predicate mentioned by Rule $\mathcal{R}1$. Such a predicate should specify the certificates an agent must present in order to perform a given operation. For example, in the Chinese Wall scenario, `authorize` should validate `test&revoke` requests made by duly certified database servers.

6 On the Implementation and Efficiency of CMP-Engines

CMP-engines are written mostly in Java. In order to avoid race conditions, requests are evaluated *sequentially* in chronological order of their arrival. As a benchmark for CMP-engine performance we measured the average time taken to compute and carry out the ruling for a given request. (Specifically, we have measured the time to evaluate a request to revoke a certificate.) The measured processing time, averaged over 3000 runs, is 3.3 ms. In the experiment the CMP-engine was running on a SUNW, Ultra-2 machine operating at 296 MHz, using Solaris 2.6 operating system and Java 1.2.

The current implementation of the engine is experimental and much less efficient than it can be. But even in its present state, it is quite affordable: the overhead due to rules evaluation is much smaller than communication time (typically, 40 ms in a WAN).

We expect future versions of CMP-engines to be much faster with the advent of better implementation of Java core libraries. In particular, it has been shown in [10] that excessive synchronization performed by low level classes can lead to a substantial performance degradation. For example, some of the I/O classes are synchronized. Since each such call requires a lock acquisition, the CMP-engine is spending much of its time acquiring locks for

<p>$\mathcal{R}1$. <code>request([Requester,LCert],R,RS) :- authorized(Requester,LCert,R), if (R = ..[Op,Cert] and processing(Op',Cert)@CS) then addCS(blocked(Requester,R,Cert)) else process(Requester,R,RS).</code></p> <p><i>When a valid request R, to perform operation Op on certificate Cert is intercepted, the rule checks if the revocation server is currently processing another operation Op' on Cert. If this is the case, the processing of R is blocked, and a term blocked(Requester,R,Cert) is added to the control state. Otherwise the request is processed (see Rules $\mathcal{R}2$ and $\mathcal{R}3$)</i></p> <p>$\mathcal{R}2$. <code>process(Requester,test&revoke(Cert),RS) :- addCS(processing(test&revoke,Cert)), deliver(Requester,test(Cert),RS).</code></p> <p><i>A test&revoke(Cert) request is processed as follows: (1) a term processing(test&revoke,Cert) is added to the control state, and (2) a request to test the status of Cert is delivered to the revocation server.</i></p> <p>$\mathcal{R}3$. <code>process(Requester,test(Cert),RS) :- addCS(processing(test,Cert)), deliver(Requester,test(Cert),RS).</code></p> <p><i>A test(Cert) request is processed as follows: (1) a term processing(test,Cert) is added to the control state, and (2) a test(Cert) request is delivered to the revocation server.</i></p> <p>$\mathcal{R}4$. <code>reply(RS,status(S,Cert),Requester) :- deliver(RS,status(S,Cert),Requester), if (S=valid and processing(test&revoke,Cert)@CS) then deliver(Requester,revoke(Cert),RS) else resume(Cert,RS).</code></p> <p><i>The response to a test(Cert) request is delivered to the Requester. Additionally, if the test-ing was triggered by a test&revoke request (i.e. the term processing(test&revoke,Cert) is present in the control state) then a revoke(Cert) request is delivered to the revocation server. Otherwise, a previously blocked request regarding Cert, if any, is resumed (see Rule $\mathcal{R}6$).</i></p> <p>$\mathcal{R}5$. <code>reply(RS,revoked(Cert),Requester) :- resume(Cert,RS).</code></p> <p><i>When a revoked(Cert) reply is intercepted, signaling that a test&revoke(Cert) operation has been carried out, then a previously blocked request regarding Cert, if any, is resumed (see Rule $\mathcal{R}6$).</i></p> <p>$\mathcal{R}6$. <code>resume(Cert,RS) :- processing(Op,Cert)@CS, delCS(processing(Op,Cert)), if blocked(Requester,R,Cert)@CS then (delCS(blocked(Requester,R,Cert)),process(Requester,R,RS)).</code></p> <p><i>When revocation server RS, finishes processing operation Op on Cert the term processing(Op,Cert) is deleted from the control state. Furthermore, the rule checks if the control state contains a term blocked(Requester,R,Cert). If such a term is found, then it is removed from the control state and the request R is processed (see Rules $\mathcal{R}2$ and $\mathcal{R}3$).</i></p>
--

Figure 6: The CMP implementing the test&revoke operation (fragment).

objects that are (usually) used by only one thread. This situation can be easily solved by providing both synchronized and unsynchronized versions for different classes. Another cause of performance degradation is excessive heap allocation [8, 17] which in turn leads to more time spent doing garbage collection.

7 Related Work

The only other framework, we are aware of, that considers the issue of regulated certificate management is Oasis [9, 13]. In Oasis a service may create certificates, which are used to authorize access to the resources managed by that service, or by other services. There are several distinctions between Oasis and the mechanism presented here. First, while Oasis supports policies regulating certificate creation of comparable sophistication to ours, the range of revocation policies is far narrower. In Oasis policies deal only with propagation of revoked certificates; their policies do not attempt to regulate the (more general) case of who is allowed to revoke a certificate, and in what conditions. For example, in Oasis, a certificate delegating a role to an agent may be revoked only by its issuer. Secondly, in Oasis certificates are to be used only by Oasis-aware services and cannot be used by other entities. Finally, since off-the-shelf PKIs are not Oasis-servers, this framework cannot be used for extending PKI functionality, nor is this issue considered.

8 Conclusion

Creation and revocation of digital certificates is one of the most complex aspects of distributed access-control. Usually certificate management is performed manually, by trusted agents, and it is rarely formalized. We contend that such an approach is expensive, error-prone and might impose a high degree of strain on a system. To deal with these problems we argue that creation and revocation of certificates should be formalized, and enforced by a general mechanism.

This paper proposes a formulation for certificate management policies and a framework for their support. In this framework, certificate management policies are enforced by generic policy engines, wrapped around certification authorities and revocation servers. The proposed framework has several important benefits. First, the language proposed here for expressing certificate management regulations is expressive enough to support a wide category of policies. Moreover, the experimental results have shown that this formulation lends itself to efficient enforcement. Second, the mechanism is easy to deploy, requiring no modifications of PK-servers. Finally, wrapping PK-servers with programmable, generic policy engines makes possible to extend the functionality of these servers, in an easy, unobtrusive manner.

References

- [1] AT&T access certificate for electronic services (ACES). <http://www.aces.att.com/>.
- [2] M. Blaze, J. Feigenbaum, and A. D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In *Security Protocols Workshop*, pages 59–63, 1998.
- [3] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 1996.
- [4] D. Brewer and M. Nash. The Chinese Wall security policy. In *Proceedings of the IEEE Symposium in Security and Privacy*, pages 206–211, Oakland, California, 1989. IEEE Computer Society.
- [5] D. Clarke, J. Elien, C. Ellison, M. Fredette, and A. Morcos. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.
- [6] C. Ellison. The nature of a usable PKI. *Computer Networks*, (31):823–830, November 1999.
- [7] D. Ferraiolo, J. Barkley, and R. Kuhn. A role based access control model and reference implementation within a corporate internet. *ACM Transactions on Information and System Security*, 2(1), February 1999.
- [8] T.H. Halfhill. How to soup up Java. *Byte*, May 1998.
- [9] R.J. Hayton, J.M. Bacon, and K. Moody. Access control in an open distributed environment. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1998.
- [10] A. Heydon and M. Najork. Performance limitations of the Java core libraries. In *Proceedings of the ACM 1999 Java Grande Conference*, June 1999.
- [11] J. Iliadis, Spinellis. D., D. Gritzalis, B. Preneel, and S. Katsikas. Evaluating certificate status information mechanisms. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 1–8, November 2000.
- [12] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a distributed firewall. In *ACM Conference on Computer and Communications Security*, pages 190–199, 2000.
- [13] K. Moody J. Bacon and W. Yao. A model of Oasis role-based access control and its support for active security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, November 2002.
- [14] J. Jajodia, P. Samarati, V. S. Subramanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *Proceedings ACM SIGMOD Conference on Management of Data*, May 1997.

- [15] G. Karjoth. Access control with IBM Tivoli access manager. *ACM Transactions on Information and System Security*, 6(2):232–257, 2003.
- [16] S. Kent. Internet privacy enhanced mail. *Communications of the ACM*, August 1993.
- [17] R. Klemm. Practical guideline for boosting Java server performance. In *Proceedings of the ACM 1999 Java Grande Conference*, June 1999.
- [18] H. Ludwig, L. O'Connor, and S. Kramer. Miera - method for inter-enterprise role-based authorization. In *Proceedings of the Conference on Electronic Commerce and Web Technologies*, pages 133 – 144, 2000.
- [19] N.H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *TOSEM, ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, July 2000.
- [20] Netscape certificate management system. <http://wp.netscape.com/cms/v4.0/>.
- [21] R.L. Rivest. Can we eliminate revocation lists? In *Proceedings of Financial Cryptography*, 1998.
- [22] RSA Keon: Certificate Authority.
<http://www.rsasecurity.com/products/keon/certificateauth.html>.
- [23] S. Stubblebine. Recent-secure authentication: Enforcing revocation in distributed systems. In *Proceedings of the 1995 IEEE Symposium on Research in Security and Privacy, Oakland*, pages 224–234, May 1995.
- [24] R. Wright, P. Lincoln, and J. Millen. Efficient fault-tolerant certificate revocation. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, November 2000.
- [25] R. Wright, P. Lincoln, and J. Millen. Depender graphs: A method of fault-tolerant certificate distribution. *Journal of Computer Security*, 9(4), November 2001.
- [26] P. R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.