# An Open Architecture for Secure Interworking Services

Richard Hayton and Ken Moody
{Richard.Hayton, Ken.Moody}@cl.cam.ac.uk
University of Cambridge Computer Laboratory
Pembroke Street, Cambridge CB2 3QG, United Kingdom

**Abstract**

There is a developing need for applications and distributed services to cooperate or *inter-operate*. Current mechanisms can hide the heterogeneity of host operating systems and abstract the issues of distribution and object location. However, in order for systems to inter-operate *securely* there must also be ways to hide differences in security policies, or at least to support negotiation between them.

Other proposals for the interworking of security mechanisms have focussed on the *enforcement* of access policy at the expense of *flexibility of expression* of policy. This work describes a new architectural approach to security. The key idea is that a *process* is the universal client entity; a process may act on behalf of an identified individual as in traditional security schemes. More generally, a process may adopt an *application-specific name* or *role*, and this is used as the basis for authentication in Oasis. A service may then be written in terms of service-specific categories of clients, decoupled from the mechanisms used to specify and enforce access control policy.

This approach allows great flexibility when integrating a number of services, and reduces the mismatch of policies that is common in heterogeneous systems. In addition, Oasis services may be integrated with alternative authentication and access control schemes, providing a truly open architecture.

A flexible security definition is meaningless if not backed by a robust and efficient implementation. Oasis has been fully implemented, and is inherently distributed and scalable. In this paper we describe the general approach then concentrate on revocation, where security designs are most often criticised. Oasis is unique in supporting the rapid and selective revocation of privileges which can cascade between services and organisations.

## 1 Introduction

In a widely distributed environment there will be many different organisations. If users and services in these organisations are to interwork, there must be standard mechanisms for the specification and enforcement of access control policies.

When a request is made of a service, that service must decide on some basis whether to perform the request. The flexibility of an access control architecture hinges on the amount of information available to the service when making this decision; if no information is provided then no useful distinction can be made between requests. Traditional approaches to security are closed in the sense that the information available is strictly limited and defined by a generic security architecture.

Ideally, we should be able to develop applications and services in terms of service-specific categories of clients. For example a Meeting application could be developed with client categories Chair, Speaker and Member. The conditions under which a particular user may assume one of these roles are a feature of the environment in which the application is used, not a feature of the application itself.

Such a scheme would provide a flexible mechanism to allow the integration of separately developed services.

Given such a scheme, care must be taken to control the complexity. In traditional schemes access control decisions are based on the identity of the user responsible for a request; typically, an access control list (ACL) specifies which principals have what access. Although inflexible, policy statements such as ACLs are easily understood and easy to manage. If more general information is available it may change rapidly and the circumstances in which a request may or may not be granted can become unclear. It is therefore essential that policies can be expressed in a clear and unambiguous way, so that contradictions and security loopholes can be discovered.[1]

Oasis is an architecture for access control that allows policies to be defined in terms of axioms in a proof system. Processes apply these axioms to prove their eligibility to enter a set of *roles*. Each service restricts its operations to clients who have proved their membership of an appropriate role. For example, a process must prove that it represents a member of a meeting before the Meeting application will allow it to invoke the `speak` operation. These proofs are generated dynamically and may be invalidated rapidly if their premises change.

Oasis is a distributed architecture. Policy is not administered centrally and there is no global name space for roles: any service may define new roles and provide policy statements describing how the roles interact with roles provided by other services. This is significant in that it allows interworking between separate administrative domains which have limited trust in each other.

Typically, each service will define one or more roles and (implicit) access control policy will be defined in terms of them. The conditions for entry to these roles can then be defined later by an administrator. The scheme trivially subsumes the traditional identity based schemes which may be specified as "the client process must represent a logged-on user on an ACL"

This paper is in two parts. In the first, we consider how general access control policies may be represented, and how policy for one service may be defined in terms of roles relating to another. In the second we consider the implementation of this scheme, with particular attention to the implementation of rapid, selective revocation of role membership (and hence access rights). A fuller description of the Oasis architecture can be found in [2].

## 2 Policies and Proofs

Access control is about determining if a request from a *process* is to be honoured. Requests do not come directly from human users, although it may be appropriate to express access control policy as if they did. For example, consider a file `/docs/Oasis.ps` protected by the ACL

$$\text{“ rjh21(}\mathtt{rw}\text{), staff(}\mathtt{r}\text{) ”}$$

If a process P representing a user `tjm15` attempts to read this file, then the file service will consult the ACL. This is equivalent to attempting to prove

P may read `/docs/Oasis.ps`

given the statement

$\vdash$ P represents `tjm15`

and the axioms

$\vdash x$ represents `rjh21` $\Rightarrow x$ may read `/docs/Oasis.ps`
$\vdash x$ represents `rjh21` $\Rightarrow x$ may write `/docs/Oasis.ps`
$\vdash x$ represents $u \wedge u$ in `staff` $\Rightarrow x$ may read `/docs/Oasis.ps`

In traditional approaches, proving the statement

$$\vdash \text{P represents } \mathtt{tjm15}$$

is considered as authentication, and this is performed by an authentication service. However there is no fundamental reason why we cannot have an ACL granting access to processes based on the machine they are running on, the program being executed or the time of day. All that is required is that a suitable statement is available during the access check and that that the file service believes that the statement is valid[3]. This is the basic mechanism used in Oasis. Client processes obtain certified statements from services and use them as credentials when accessing other services. To manage complexity, these statements are of a restricted form and represent role memberships, as described in the following section.

# 3 Role Membership Certificates

A role membership certificate is a certified statement that a particular process may represent a particular role. For example

- Process `P` represents the user `tjm15` *within the Computer Lab.*
- Process `Q` represents a member of the meeting *within Meeting 123*
- Process `R` represents a client who may read file `/docs/Oasis.ps` *in the* CL *filing system.*

Each of these statements indicates that the process is a member of a particular role. In the first example the role is **User**(`tjm15`), in the second **Member**() and in the third **UseFile**(`/docs/Oasis.ps`, `read`). Note that representing a user is a role like any other. We do not have 'user in role' semantics. Roles are parametrised as this simplifies policy definition.

The italicised part of the statement represents the context in which it is valid. For example only the Computer Laboratory login service may issue statements about which processes represent Computer Lab. users. Role membership certificates are implemented as identity based capabilities, and are protected by a secret known only to the issuing service. In this way, context is managed implicitly.

Any client who has obtained a role membership certificate may approach a service and attempt to gain access. The service may simply use this certificate (for example the file service may expect **UseFile**() certificates) and this gives capability-like semantics. Alternatively, a service may apply further axioms, for example by consulting an ACL.

Note that the service does not care how the process came by the certificate used in the access check; it is simply concerned with its validity. This allows us to decouple the implementation of a service from the management of access control policy.

# 4 Specifying Policy

Policy is specified by providing axioms for use by potential clients wishing to enter a role. These axioms take the form of statements in a role definition language (RDL). For example a meeting application may provide two roles: **Chair** and **Member** and have role definitions as follows:

$$\begin{array}{lll} \textbf{Chair} & \leftarrow \textbf{Login.User}(\texttt{jmb}) & \\ \textbf{Member}(u) & \leftarrow \textbf{Login.User}(u) \triangleleft \textbf{Chair} & : u \text{ in } \texttt{staff} \end{array}$$

The first statement indicates that a client with the role **User**(`jmb`) may enter the role **Chair**. As any number of services may issue **User** certificates, the policy is incomplete if we do not specify which service these certificates should have been issued by. In this example the name 'Login' is looked up in a trusted name server and returns the address of the system login service.

The second statement is used by processes wishing to enter the `Member`() role. It states that a process must hold a suitable login certificate and be delegated by the Chair. It is equivalent to the following axiom:

$$\frac{\begin{array}{c} c \text{ represents } \textbf{Chair} \\ c \text{ wishes to delegate } \textbf{Member}(u) \text{ to } c' \\ c' \text{ represents } \textbf{Login.User}(u) \\ u \text{ in } \texttt{staff} \end{array}}{c' \text{ represents } \textbf{Member}(u)}$$

Generally, a Chairperson will think of delegation as referring to a person not a process. A delegation request by the chair may be of the form 'delegate **Member**(rjh21) to **User**(rjh21)'. A request of this form by a process $c$ will result in a delegation certificate representing the following axiom being returned.

$$\frac{c' \text{ represents } \mathbf{Login.User}(\texttt{rjh21})}{c \text{ wishes to delegate } \mathbf{Member}(\texttt{rjh21}) \text{ to } c'}$$

This may be passed to a potential member and, together with a role membership certificate for the role **User**, may be provided as a credential when attempting to enter the role **Member**.

# 5 Specifying Revocation

In the above sections we have described how flexible access control policies may be specified as statements in a role definition language that correspond to axioms in a proof system. However, in order for this system to be of use, we must be able to withdraw statements that no longer hold. For example, if the Chair wishes to revoke the membership delegated to rjh21, this may be considered as a withdrawal of the statement

$$c \text{ wishes to delegate } \mathbf{Member}(\texttt{rjh21}) \text{ to } c'$$

In order to allow freedom of policy expression, it must be possible to give statements relating both to the conditions required for role entry, and to the conditions under which that entry should be revoked. In RDL this is done by annotating the statements to indicate which phrases represent *entry conditions* and which represent *membership rules*. An entry condition need only hold at the time of role entry, but membership rules must remain valid. If at any time after role entry, a role membership cannot be proved using axioms formed by the membership rules related to it, then the membership is revoked. Membership rules are indicated by appending an asterisk. For example the statement

$$\mathbf{Member}(u) \quad \leftarrow \mathbf{Login.User}(u) \triangleleft^{*} \mathbf{Chair} \quad : (u \text{ in } \texttt{staff})^{*}$$

Indicates that Membership will be revoked if the Chair reverses the revocation decision (as $\triangleleft$ is a membership rule) or if the user represented by the membership ceases to be a member of the group staff (as $u$ in staff is a membership rule). Nothing else will affect the role membership. Efficient implementation of revocation in Oasis proved to be the key issue for its overall performance, and an effective mechanism is described in the following sections.

# 6 Validating Certificates

Suppose that a client supplies a certificate as a credential, either when performing an operation, or when entering a role; we must validate that the certificate is genuine, was issued to the client making the request and has not been revoked. There are three stages to validation. Firstly, the client identifier is validated using a suitable authentication protocol. This ensures that one process cannot masquerade as another. Secondly, the integrity of the certificate is validated by recomputation of a digital signature. Finally, a field within the certificate is used to determine if the certificate has been revoked.

The digital signature is a function of the certificate text, the process identifier and the issuing service. This is based on that used in [4] and ensures that a certificate cannot be 'stolen' or used out of context. Unlike other schemes, in Oasis the *only* function of the digital signature is to check that the signature is not forged. Once the check has been performed, the integrity of the certificate may be cached, and re-computation avoided. This is particularly significant in distributed architectures where validation may involve a remote procedure call to the issuing service.

In addition, if the digital signature check fails, then the service can be certain that the certificate has been modified. If the messages are protected from accidental corruption by checksums, then a failed signature check is a good indication of an attempt by a client to gain illicit access. This is a great improvement over schemes in which signature checks frequently fail for a number of reasons, making attempted fraud difficult to detect.

# 7 Credential Records

Credential records allow flexible, selective revocation due to any number of 'significant events'. A credential record is a small record stored in a server that represents that server's current belief about some fact. Records form a directed graph, such that a child represents some function of the beliefs held about its parents. In this way, only a single credential record need be consulted to confirm an arbitrary number of facts. A field is added to each certificate called a 'credential record reference'. This is a reference to a record within the issuing server that represents the validity of the certificate. The name space for credential record references is designed so that references are never reused, and credential records representing facts that are false, and will always remain false, can be deleted.

# 8 Constructing Credential Record Graphs

Graphs of credential records correspond directly to proofs formed by the instantiation of RDL axioms. Each membership rule involved in the definition of a role will be represented by a single credential record. Consider the definition of 'Member' from the previous section.

$$\mathbf{Member}(u) \quad \leftarrow \mathbf{Login.User}(u)^{*} \lhd^{*} \mathbf{Chair} \quad : (u \text{ in } \texttt{staff})^{*}$$

There are three membership rules for this definition:

1. The supplied logged on certificate must remain valid. The certificate contains a reference to the credential record representing this fact.
2. The delegation must not be revoked. A new credential record is created to represent this fact. The delegator is given the right to delete this record.
3. The client must remain a member of the group 'staff'. Each group membership is represented by a single credential record, and membership lookup returns a reference to this as a side-effect.

To create a suitable credential record to represent the truth of all three of these facts one new record must be created (for rule 2) and a second record must be created to 'and' together the records representing rules 1, 2, and 3. A small optimisation is possible in that the two new records can be combined into one fulfilling both functions. In general one new credential record is required for each (revokable) delegation, and one for each entry to a role with multiple membership rules.

# 9 Distribution Issues

In a distributed environment, certificates issued at one server may be used as credentials at another. Consequently, a credential record in one server may be required to be the parent of a record in another. This raises issues of naming, independent failure modes and robustness. In order to decouple the name space and failure modes of two services, *external records* are used to represent remote facts and *event notification* is used to communicate state changes between servers.

## 9.1 External Records

If a server requires a reference to a credential record on another service, it creates a local surrogate record called an *external record*. This record contains information about the identity, location and state of the record being represented, together with the standard attributes of a credential record, including an identifier within the local name space. The state of the record is maintained by event notification, as described below, and in all other respects the record is treated as a local credential record.

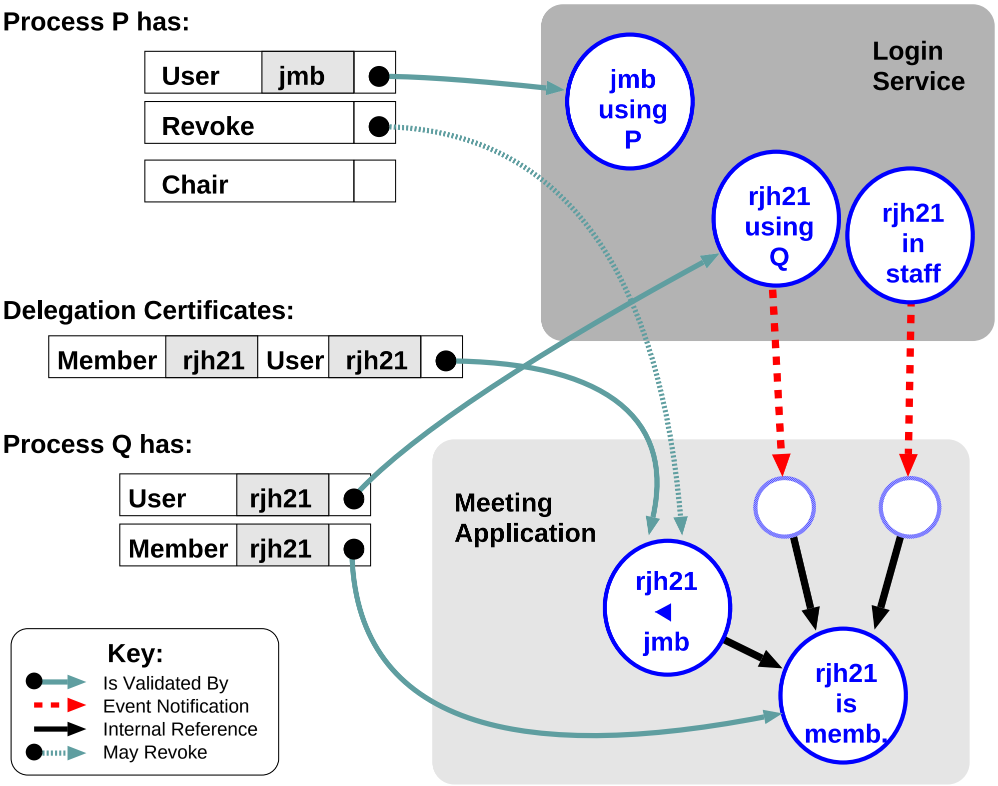


Figure 1: A credential record Graph

## 9.2 Event Notification

Asynchronous event notification is an important feature of distributed systems, and the RPC mechanism used in the current implementation of Oasis has been extended to add event management functions[1]. Oasis makes use of these functions by defining the event type **Modified**(*CRR*, *newstate*) in the interface definition file of an Oasis server. A server may then register interest in the state of a particular credential record, and will be informed if its state changes, by being sent an event with *CRR* and *newstate* set to appropriate values.

In this way, revocation taking place in one server may affect certificates issued by another. Revocation is therefore both rapid and selective. In addition, as revocation is event-based rather than depending on timeouts, there is a low 'background' overhead to using certificates, since they do not need to be continually refreshed. Figure 1 gives a snapshot of the certificates issued relating to a meeting involving two client processes. Process `P` represents a logged in user `jmb` who has become the chair of the meeting. `jmb` has then delegated a user `rjh21` to be a member. A process `Q` representing this user has accepted the delegation and has obtained a **Member** certificate. This certificate may be validated by ensuring that the credential record it references has a known value and is true. If any of the membership rules fail (for example `rjh21` is ejected from the group `staff`) then the credential record will be invalidated, and the Membership certificate will no longer be accepted.

Note that the Chair certificate itself can never be revoked, as no membership rules were given in the definition of Chair. Of course, the certificate is only usable in one instance of the meeting service and the Chair certificate is useless once the meeting finishes.

[1] This is described in a companion paper at this workshop [5]

## 10 The Effect of Failures

Event notification between servers may be delayed indefinitely by network congestion or failure. Additionally, either of the parties may fail and restart independently. These situations must be taken into account in the design of any distributed system involving events. The approaches described here are implemented as part of a generic event library, and are equally admissible to any event-based application.

Consider two parties `A` and `B`, where `A` wishes to send `B` a stream of messages. If every message `A` sends contains a sequence number, then `B` will be able to detect if any *previous* message has been lost. If in addition, `A` ensures that a message is sent at least every $t$ seconds, then `B` will know within time $t$ if a message has been lost or delayed.

This is the basic requirement for event handshaking. In addition, `B` must periodically inform `A` that events have been received, so that `A` may delete any associated state.

This protocol is called a *heartbeat protocol*, and a form of it is used in the event system implemented. The server responsible for signalling events is used as the initiator of the protocol and ensures that a heartbeat event is sent every $t$ seconds. Individual events (and heartbeats) are not acknowledged for reasons of efficiency, but the client replies periodically, so that the server can detect failures and resend event instances if required.

This leads to a system with the following characteristics:

- A client can be certain of receiving an event within time $t$ of its generation, or of detecting that notification may have failed or been delayed.
- A server can detect a client that is not responding, and after a period can assume that it is no longer running.
- A client who processes and forwards events can treat heart-beats in a similar manner. This feature allows a service to provide guarantees about 'indirect' events from other services.

In Oasis, missed heartbeats lead to external credential records being marked as 'unknown'. This state propagates to child records, and possibly to other servers. While the state of a record is unknown, certificates relying on it may not be used. When connection is re-established the state of each record is read and, if necessary, events are re-registered with the remote service. The period of the heartbeat is negotiated so that if, for example, group membership changes are rare and revocation due to these changes may be delayed, then the heartbeat period may be set to once every few hours rather than once every few seconds. Of course, regardless of the heartbeat, revocation will still be rapid if there are no communications failures.

## 11 Relaxing the Rules

As described above, in order to validate an Oasis certificate, the issuing server must be consulted. This is because only the issuing server will know whether the certificate has been revoked, and only that server will be able to inform a client application should it be revoked in future. In widely distributed systems this cost may be unacceptable. If we are to improve the performance we must sacrifice something of the semantics. We can create a second form of certificate protected by public key signatures. These can be validated without recourse to the issuing service, but revocation information will not be available. Instead, each certificate can contain timestamps indicating when it is valid. Following these periods the certificate is discarded so that revocation is effectively delayed by a period equal to the lifetime of the certificate. By combining the two forms of certificate, a system designer can fine tune the performance trade-offs. Alternative schemes that rely only on timed-out certificates do not provide this flexibility. We envisage that credential-based certificates will be the norm, with timed certificates being used to represent relatively stable information that is useful over a wide area, for example group or project membership.

## 12 Integration with Alternative Security Mechanisms

As described in section 3, when services make access control decisions, they rely on a number of statements in the form of certificates. As these certificates can be used to represent any role membership, a gateway can be devised to translate access control information from other security schemes. For example a service may issue **Kerberos**(*userid*) certificates, indicating that a process has been authenticated by a Kerberos server to represent the given UserId. As Oasis supports rapid revocation, the revocation schemes of other services can be mimicked.

A second issue is reasoning about security policies. If all policies are defined in terms of Oasis RDL statements, then reasoning is relatively straightforward as the policies have common semantics and a strong formal backing. When policies from other services are combined, reasoning is more difficult. However, as RDL is an expressive language, it is often the case that alternative policies may be converted to RDL. Even if this does not aid implementation, it greatly simplifies reasoning about interacting policies. See [2, chapter 3] for a detailed treatment of the conversion of UNIX ACLs to RDL statements.

## 13 Conclusions

There are two primary access control issues; the specification of access policy and the enforcement of that policy. In a distributed environment the specification mechanism must be extremely flexible, as different organisations have radically different needs. Unlike other proposals for distributed access control, Oasis has concentrated on the specification of policy, and how policies interact, rather than on the mechanisms. We believe the result is a clean and simple architecture that hides heterogeneity and so aids reasoning. Far from being inefficient, we have found that that the distributed proofs can be represented by an extremely efficient mechanism: credential records. This is a general mechanism for the representation of beliefs, and may be used to allow Oasis services to interwork with other security mechanisms, as well as providing the framework for the Oasis implementation itself.

## Acknowledgements

We acknowledge the UK EPSRC for supporting this work under grant GR/K77068. We are grateful to ICL for general support of our research group.

## References

[1] Morris Sloman, Policy Driven Management for Distributed Systems, In *Journal of Network and Systems Management, Plenum Press*, 2(4) 1994

[2] R. Hayton, OASIS, An Open Architecture for Secure Interworking Services, University of Cambridge PhD thesis, Technical Report 399. 1996.

[3] Butler Lampson, Martin Abadi, Michael Burrows and Edward Wobber, A Calculus for Access-Control in Distributed Systems, In *ACM Transactions on Programming Languages and Systems*, 15(4):706-734, 1993

[4] Li Gong. A secure identity-based capability system. In *Proceedings of the 1989 Symposium on Security and Privacy*, pages 56–63. IEEE, May 1989.

[5] R. Hayton, J. Bacon, John Bates, and K. Moody, Using Events to Build Large Scale Distributed Applications, ACM SIGOPS European Workshop, September 96.