dRBAC: Distributed Role-based Access Control for Dynamic Coalition Environments (TR2001-819)

Eric Freudenthal, Tracy Pesin, Lawrence Port,
Edward Keenan, and Vijay Karamcheti
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
{*freudent, tracyp, lport, woodiek, vijayk*}*@cs.nyu.edu*

Abstract

Distributed Role-Based Access Control (dRBAC) is a scalable, decentralized trust-management and access-control mechanism for systems that span multiple administrative domains. dRBAC represents controlled actions in terms of *roles*, which are defined within the trust domain of one entity and can be transitively delegated to other roles within a different trust domain. dRBAC utilizes PKI to identify all entities engaged in trust-sensitive operations and to validate delegation certificates. The mapping of roles to authorized name spaces obviates the need to identify additional policy roots.

dRBAC distinguishes itself from previous trust management and role-based access control approaches in its support for three features: (1) *third-party delegations*, which improve expressiveness by allowing an entity to delegate roles outside its namespace when authorized by an explicit *delegation of assignment*; (2) *valued attributes*, which modulate transferred access rights via mechanisms that assign and manipulate numerical values associated with roles; and (3) *credential subscriptions*, which enable continuous monitoring of established trust relationships using a pub/sub infrastructure to track the status of revocable credentials.

This paper describes the dRBAC model, its scalable implementation using a graph-based model of credential discovery and validation, and its application in a larger security context.

1 Introduction

dRBAC was motivated by the problem of controlling access to resources in *coalition environments*. A “coalition environment” could be military, in which several nations work together to achieve a common goal; or commercial, in which corporations form a partnership. The defining characteristic of a coalition environment is the presence of multiple organizations or entities that have no common trusted root authority. In such a situation, the entities must cooperate to share the subset of their protected resources that is necessary to the coalition, while protecting the resources that they don’t want to share. The growth of network-based services on the Internet promises to make this paradigm pervasive. For instance, one can imagine a situation where a mobile user seamlessly obtains connectivity to the Internet using the network facilities at a hotel or an airport based on a coalition between the user’s organization and the entities administering these networks. Upon receiving the user’s request, the network gateway can determine the existence of a trust relationship between its organization and the user’s, and if one exists, enable access at the specified level *in spite* of the anonymous and transient nature of the user’s relationship with the network.

Although such scenarios are compelling, establishing trust in coalition environments presents a number of challenges:

- The members of the coalition must securely identify themselves to each other, possibly by transmitting information over insecure network links. The identities of coalition members might not be known to each other a priori, requiring the members to prove their trustworthiness through the presentation of additional evidence. (This problem is familiar to users of *ssh* [20], for example, who must make an initial decision as to whether to accept the key of an unknown host.)

- Coalitions involving large organizations benefit from an access control system that recognizes and maintains the natural structure of the organization. It should be possible to authorize a user based on some "role" that the user performs within an organization.

- Highly dynamic coalition environments must support the transitive delegation of authorization. The owner of a resource must not only be able to specify who can access the resource, he must also be able to identify users who can further delegate that authorization to others. This should be accomplished in a way that minimizes the amount of coordination necessary between the owner of the resource and the entity who can further delegate it. The evidence that establishes a transitive delegation of authority to a user could conceivably be distributed over a number of different network hosts. Such evidence will ideally be collected and presented automatically.

- Established trust relationships must be continuously monitored to track the status of revocable credentials. This is particularly crucial in the case of prolonged user accesses to some secure resource.

Unfortunately, existing solutions do not address these challenges well. Simple access control lists pose administrative difficulties, don't scale well to large systems and do not permit the transitive delegation of authority. Traditional role-based access control (RBAC) systems [18, 19] depend upon a central trusted computing base administered by a single authority, which contains all of the security policy for the entire organization. RBAC systems provides ease of administration: when a user is added to a group, that user is instantly afforded all the rights and privileges of that group. However, this approach cannot be scaled for large number of mutually anonymous users such as one might encounter in coalition settings. More recently, trust-based systems (e.g., SDSI/SPKI [17, 7], Keynote [2, 3]) have been developed to control access. The application controlling the resource uses public-key cryptographic signatures to authenticate the individuals requesting access. This approach has the advantages of decentralized administration, scalability, and the ability to authenticate individuals over insecure networks. However, most trust-based systems do not provide a delegation language that mirrors the natural structure of organizations. They also do not address the question of continuously monitoring established trust relationships for changes to credential status.

Our approach, dRBAC, combines the advantages of role-based access control and trust-management systems to create a system that offers both administrative ease and a decentralized, scalable implementation. dRBAC represents controlled actions in terms of *roles*, which are defined within the trust domain of one entity and can be transitively delegated to other roles within a different trust domain. dRBAC utilizes PKI to identify all entities engaged in trust-sensitive operations and to validate delegation certificates. The mapping of roles to authorized name spaces obviates the need to identify additional policy roots. This paper describes the key components of dRBAC: its model for authorizing actions, and a scalable architecture to support distribution, discovery, validation, and monitoring of credential chains.

dRBAC has been influenced by the design of other recently proposed trust management systems, most notably RT0 [13] whose delegation chain discovery ideas have informed dRBAC's algorithms. However, dRBAC distinguishes itself from previous approaches in its support for three novel features:

- *Third-Party Delegations*, which allow an authorized entity to delegate roles that were created by *another* entity by referring directly to the role originator's namespace. This mechanism, related to the *speaks for* relationship of Li's Delegation Logic [12], allows a natural administration model where privileged entities in the system can create roles and designate other (less privileged) entities to give out these roles. More importantly, as we show in Section 3, third-party delegation actually increases the expressiveness of dRBAC in important ways. These benefits come at little cost: third-party delegation does not require any additional infrastructure or specification of system security policy beyond signed delegations.

- *Valued Attributes*, which take on numerical values to capture different levels of access and enable *fine-grained* rights restriction via mechanisms that modulate access rights transferred through delegations. Valued attributes, like roles, are created by a given entity and defined within that entity's namespace. As with third-party delegations, valued attributes are specified through the issuance of signed delegations, and require no out-of-band policy.
- *Credential Subscriptions*, which enable continuous monitoring of established trust relationships using a pub/sub infrastructure to track the status of revocable credentials. This feature allows the dRBAC infrastructure to support prolonged interactions based upon trust establishment at setup time. The dRBAC system *pushes* updates to client applications on the status of critical delegations, enabling propagation of revocation information in a more efficient fashion than the alternative that requires constant polling.

These three features of dRBAC enable the construction of a powerful trust management and access control system. Like other trust management systems (see Blaze et. al. [2]), mechanisms that provide access are separated from policy. No globally trusted 'certifying authority' is required: Each authority responsible for protected resource can define their own trust relationships with other entities throughout the distributed system. Privilege to access restricted resources is completely specified through the issuance of delegation credentials that describe trust relationships among partners. Mechanisms are provided to automatically discover chains of delegations authorizing a required trust relationship. Finally, mechanisms are provided to monitor the status of a trust relationship over the duration of a sustained transaction. To understand the benefits that accrue from these features, consider the following example.

Example Consider the problem of dynamically distributing a video data feed to members of a military coalition involving three countries, say the United States (US), United Kingdom (UK), and Australia. Imagine that US cameras capture video information and want to transmit it digitally to not just US troops, but also British and Australian troops that are engaged in a certain military exercise. Imagine also that these troops may be connecting to the video source via computers that are unknown to machine that is hosting the video feed, and over insecure network links. In addition, the US would like to modulate the delivery of the data feed such that American officers receive a higher image resolution than other users.

Using dRBAC, all of these requirements can be supported in a straight-forward manner through the issuance of delegations. Perhaps the host of the video feed specified that US Generals can both view the feed and also delegate that right to others. US Generals would therefore be able to issue delegations that defined the rights of Australian and British troops to view the feed. These delegations, enabled by dRBAC support for third-party delegation, avoids conflicts in the namespace associated with the role "US Generals" and has a clarity advantage. dRBAC support for valued attributes allow delegations issued to troops from different countries to be associated with different resolution settings achieving the desired prioritization. Finally, support for credentialed subscriptions allows the feed distribution application to be informed of when the exercise completes (as indicated by a change of credentials for non-US troops), enabling it to immediately cut off the feed for disallowed users.

The dRBAC system is one part of a larger security infrastructure called DisCo being developed by our research group. DisCo enables dynamic deployment of decomposable services in partially trusted environments relying upon dRBAC for mutual authentication and authorization based upon client and server credentials. DisCo also includes a novel abstraction called the Switchboard, which provides applications with the ability to create credentialed, secure connections with the same programming effort as might be required in a completely secure environment. The overall DisCo architecture and the Switchboard abstraction are described in additional detail elsewhere [9, 10].

The rest of this paper is organized as follows. Section 3 discusses the basic dRBAC model for delegating authority, as well as extensions that support modulation of access rights and control of credential lifetime. Section 4 presents an architecture that supports the distribution, discovery, validation and continuous monitoring of certificate chains. Section 5 contains an extended case study that shows the use of dRBAC in a larger security context. We conclude the paper with a discussion of related work and future implementation and design goals.

2 Overview of dRBAC

Fundamentally, dRBAC authorizes accesses to secure resources by ascertaining whether the requesting Principal has been granted a role that the secure resource requires for access. The key question that dRBAC attempts to answer is "Does Principal P have Role R?"

dRBAC uses the set of "building blocks" defined below to answer this question:

Credentials We refer to the relationship between a Principal and some Role that the principal has been granted as a *Credential.* If a user can act in a given Role, he must have a Credential that proves his right to do so. This term is somewhat informal; when we discuss the concrete software object that implements Credentials, we will call it a *Delegation.*

Credential Chains Credentials can be passed from principal to principal in a transitive fashion. A user U who has been granted Role R *may* be able to further delegation Role R to others, depending on the restrictions specified by the author of the credential granting R to U. The transitive passing of Roles from Principal to Principal can be imagined as a *Credential Chain*.

Supporting Chains In dRBAC, it is possible for a user U to delegate a Role R that U did not himself define. In this case, U must provide evidence that he had the right to delegate R. This evidence will also take the form of a *Credential Chain* rooted by the author of R, and is termed a *Supporting Chain*.

Proofs The full set of *Credential Chains* and *Supporting Chains* needed to answer the question "Does Principal P have Role R" is called a *Proof.* Figure 1 depicts a proof demonstrating that Principal A has the rights to a role created by B, called Role 2. If such a Proof exists, we say that $\mathtt{A} \Rightarrow \mathtt{B.Role2}$.

Sub-Proofs The concept of a sub-proof will become important when we examine the process by which dRBAC builds full Proofs from individual Delegations. A valid Proof is composed of one or more Sub-Proofs. A Sub-Proof is a continuous subset of delegations in the Primary chain along with all of the needed Support Chains, such that the Sub-Proof provides full evidence of a trust relationship between the Subject in the first delegation of the Sub-Proof's Primary Chain and the Object in the last delegation of the Sub-Proof's Primary Chain.

Proof Monitor Once a Proof that $\mathtt{A} \Rightarrow \mathtt{B.Role2}$ has been identified, it is necessary to ensure that all of the Credentials within the Proof remain valid for the duration of the access. A *Proof Monitor* provides this functionality; this object is described in more detail in Section 4.

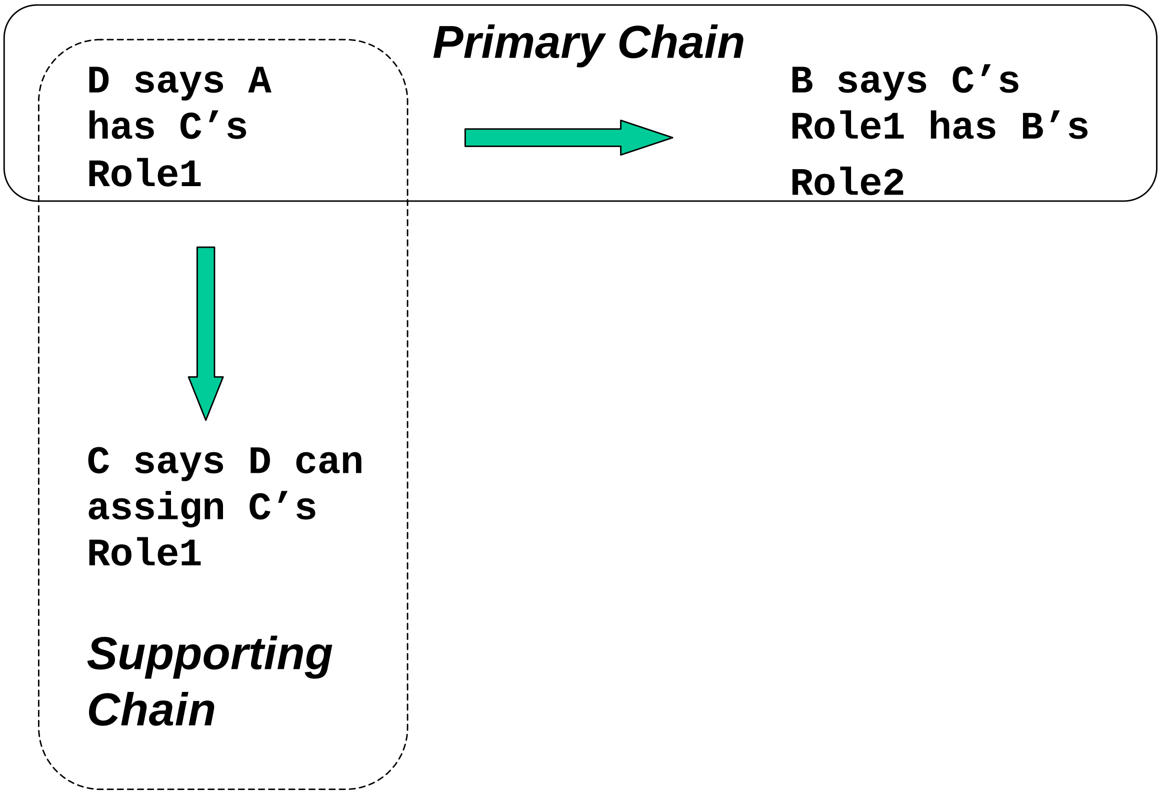

Figure 1: dRBAC Proof that A⇒B.Role2

Section 3 formalizes these notions through a precise description of dRBAC’s delegation language. Section 4 then describes an infrastructure that can be used to deploy dRBAC in a distributed setting.

3 dRBAC Model and Theory

3.1 Base model for delegations

Table 1 gives syntax and usage examples for the base dRBAC delegation model. We now describe the components of this model in more detail.

3.1.1 Entities

Unlike some other trust management systems, dRBAC does not make a hard distinction between the “resources” protected by the system and the “principals” attempting to access them. In dRBAC, all are termed “entities”, and all are represented by a unique public/private key pair. Relaxing the distinction between resources and principals allows for more transitivity in specifying delegation relationships. In particular, in our system, all users, hosts, and services are represented by entities.

All entities are mapped to a public/private key pair so that they can be uniquely identified. Entities may represent a group of individuals as well as an individual, for example, both “Cardiologist” and “Dr. Bob Smith” are dRBAC entities. See Table 1 for examples.

Entities	A public key that represents a principal or a resource, and defines a namespace that can contain roles.
Form:	cryptographic public key and a human-readable name
Examples:	`CEO`; `classifiedDatabase`
Roles	A name within an `Entity’s` namespace.
Form:	`Entity.LocalName`
Examples:	`CEO.assistant`
`secureDB.table1`	
Role Delegations	Signed Certificates that grant extend access rights on some `Object` to a `Subject`.
Form:	`[Subject → Object] Issuer` where `Object` is a `Role`, `Issuer` is an `Entity`, and `Subject` is a `Role` or an `Entity`.
dRBAC includes three major types of delegations:	
Object Delegation	An Issuer A grants role A.a to some `Subject`. The role granted is defined within A’s namespace.
Form:	`[Subject → A.a] A`

Examples:	`[Raymond → CEO.assistant] CEO`
	`[DBGroup.Developer → secureDB.table1] secureDB`

Assignment Delegation:	Entity B grants some `Subject` the right to delegate `Role` A.a to others. The tick (’) indicates that the `Subject` can further delegate the `Role`. B and A may or may not be the same `Entity`.
Form:	`[Subject → A.a’]B`

Examples:	`[HumanResources → CEO.assistant’] CEO`
	`[DBGroup.Admin → secureDB.table1’] CTO`

Third-Party Delegation:	In third-party delegation, some `Issuer` B exercises their right to delegate a `Role` defined in A’s namespace. A and B are **not** the same `Entity`.
Form:	`[Subject → A.a]B`

Examples:	`[Jen → CEO.assistant]HumanResources`
	`[Marketing → secureDB.table1] DBGroup`

Table 1: Syntax for the base dRBAC delegation model

3.1.2 Roles

The central construct in dRBAC is a “role.” Any user wishing to perform a protected action must first prove that he can act in the necessary role.

Roles are names within the namespace of a given entity. They are used to name something that the entity selectively control the access rights to, whether it be a protected resource such as a database, or a position within an organization that implies a set of rights, such as Manager.

3.1.3 Delegations

Roles are granted to subjects via “delegations.” At a high level, the format of a delegation is:

`[Subject → Object] Issuer`

where the Subject is a Role or an Entity, the Object is a Role, and the Issuer is an Entity. The arrow “→” can be read as “has the role of.” This relationship is enclosed within square brackets, and signed by the “Issuer”, which is the Entity responsible for creating (”issuing”) the delegation.

An important feature of dRBAC, and one which distinguishes it from many other systems, is that the Issuer need not be the same Entity as the Object Entity (the Entity in whose namespace the Object Role is defined.) dRBAC attempts to validate the relationship between the Object and the Issuer at the time that the delegation is used. In order for the Subject to employ the role granted in the delegation, it must be proved that the Issuer in fact had the right to give away the Object in question. The ability to grant a given role is called the “right of assignment” of that role.

Very often, the Object Entity and the Issuer will in fact be the same entity, in which case the delegation is called “self-certifying.” These delegations are valid by definition, since anyone can give away any of the roles in their own namespace. All valid dRBAC certificate chains terminate with self-certifying delegations.

Object Delegation Recall that the Object of a delegation is a Role, where a Role is a string attribute within the namespace of an Entity. It is important to note that, since the Object of a delegation must be a Role (never an Entity), an Entity cannot delegate its identity to someone else.

The Subject may be either a Role or an Entity. If the Subject is a Role, then the form of the delegation is:

`[Entity.LocalName → Entity.LocalName] Issuer`

This delegation has the effect of mapping a Role in the Object’s namespace onto a Role in the Subject’s namespace. Since any Entity can give away any role in its namespace, the Subject of this delegation will be able to further delegate the granted Role.

If the Subject is an Entity, then the form of the delegation is:

`[Entity → Entity.LocalName] Issuer`

This delegation grants the Subject Entity on the left-hand side of the arrow the Object Role indicated on the right side of the arrow.

Since in this case the Subject is an Entity and not a Role, the Object Role is not mapped to a Role in the Subject’s namespace. Therefore, in general, the Subject of this delegation will not be able to delegate this capability further. (However, dRBAC’s “third-party delegation” feature, detailed below, will make it possible for an Entity Subject to delegate roles further by referring to the Object’s namespace.)

Assignment Delegation In assignment delegation, an Entity grants some Subject the right to delegate a Role without mapping that Role into the Subject’s namespace. In dRBAC terms, the Subject is given *right of assignment* on the Object Role.

dRBAC syntax for “right of assignment” is a tick (’) after the Object. In the following delegation:

`[Subject → Object′] Issuer`

the Subject is given *right of assignment* on the Object Role. This will allow the Subject to issue valid delegations in which it gives away the Object Role by referring to the Object's namespace, instead of its own.

Third-Party Delegation In dRBAC's Third-Party Delegation, the Issuer delegates a Role that exists in another user's namespace. The validating engine must ensure that the Issuer has the *right of assignment* on the Object.

Here is a general example to illustrate the concepts:

Principal A wants to use the following delegation in order to employ Role B.b.

(1) [A→B.b] C

If the Object and the Issuer are the same entity (B=C), then the delegation is a "self-certifying" delegation. If they are not the same entity, then it must be proved that C has the right to delegate this Role of Entity B. The following delegation, presented in conjuction with the one above, would supply the necessary evidence:

(2) [C→B.b′] D

The prime (') after B.b in the delegation above indicates that the *right of assignment* on Role B.b was granted to C, such that C can now delegate Role B.b to others.

Of course, this is a recursive formulation; it must now be proved that D had rights of assignment on B.b. Each such chain must terminate in a *self-certifying delegation*, where the Object and Issuer are the same Entity, such as the following:

(3)[D→B.b′]B

Taken together, delegations (1),(2) and (3) compose a valid delegation chain granting Entity A role B.b

To recap, principal A wants to employ the role B.b. Principal A presents three certificates that say that (1) C says A has the role of B.b; (2) D says that C has right of assignment on Role B.b; and (3)B says that D has right of assignment on Role B.b.

In this way, each entity becomes its own Certification Authority (CA). The self-certifying credentials obviate the need for any external policy specifying which authorities to trust.

Benefits of Third-Party Delegation

Clarity of Delegations and Namespace Management A major objective of Third-Party Delegation construction is clarity and expressiveness. An entity can be given rights to function as a 'role assigner' rather than simply a delegator of their own rights. It may be more natural for a user to recall that they have the right to delegate CameraA.view, rather than keep track of the fact that the view role of CameraA maps to their own view role.

In addition, third-party delegation can be used to reduce namespace conflicts. If the same user, call her Alice, now wants to assign CameraB.view, and the view attribute in her namespace is already in use, she need not find a new role name to use. Instead, she can explicitly name CameraB.view, referring to CameraB's namespace, in any delegations she writes. Of course, this requires some coordination with the Issuer that gave Alice the right of assign on CameraB – they need to have used the Third-Party Delegation mode. However, notice that with Third-Party Delegation, CameraB and Alice do not need to agree on the meaning of view in their respective namespaces, whereas with Object Delegation they do.

Functional Differences In addition, Third-Party Delegation can create some functional capabilities that are absent in Object Delegation. For example, Third-Party Delegation provides an interesting means of "grouping" capabilities into a Role, and then delegating right of assignment on that Role to another user, thereby giving the user the right to delegate any or all of the capabilities that are associated with the Role. This can be of immense value in a system where a Role may encompass a large number of capabilities.

In this example, a US General is given the ability to assign three roles:

```
[US.General → Missile.Fire’] Missile
```

```
[US.General → Camera.View’] Camera
```

```
[US.General → Tank.Drive’] Tank
```

```
[Bob → US.General]US
```

Now, Bob cannot assign the role of General any further since it is not mapped to a role in his namespace. This is the desired behavior. However, since he is a `US.General`, there are valid delegation chains giving him the right of assignment on `Missile.Fire`, `Camera.View`, and `Tank.Drive`. Suppose that Bob would now like to delegate the rights to view the Camera and drive the Tank, but not the right to fire the Missile, to Private Joe. He can issue the following delegations:

```
[Joe → Camera.View] Bob
```

```
[Joe → Tank.Drive] Bob
```

Notice that these delegations do not give Joe the right to delegate these capabilities further. They are neither mapped to roles in his namespace, nor does Bob include the tick (’) granting right of assignment.

Using Object Delegation, there would have had to be individual delegations mapping each of the Missile.Fire, etc. roles into Bob’s namespace, ensuring that none of them conflicted with roles already existing in Bob’s namespace:

```
[Bob.Fire → General.Fire] General
```

```
[Bob.View → General.View] General
```

```
[Bob.Drive → General.Drive] General
```

This not only increases the number of delegations in the system (potentially by a large amount in a real system, where Subjects have many capabilities), but also increases the amount of coordination needed to avoid namespace conflicts.

3.2 Extensions to Base Delegation Model

Table 2 gives syntax and examples for the extensions to the base dRBAC model. These extensions include *Valued Attributes* and *Credential Management* information. We now describe them in more detail.

3.2.1 Valued Attributes

Often, trusted resources naturally permit accesses at varying levels of service. dRBAC supports access control specification for such resources using the notion of Valued Attributes. Valued Attributes can be used to modulate the level of access or the quality of service granted to an authorized user. Like Roles, Valued Attributes exist in the namespace of a given Entity, but they are disjoint from the set of Roles belonging to that Entity.

One or more valued attributes may be set in conjunction with the delegation of some Role. It is only meaningful to set attributes that are defined within the namespace of the Object Role, or that are inherited by that Object Role.

Also like Roles, the right to delegate Valued Attributes can be assigned to third parties. For example, consider a live video feed `Camera.view`. An `Officer.intelligence` should receive the data as close to real-time as possible. However, there may be a security need to delay the delivery of data to

Valued Attributes	A name within an Entity’s namespace that can be set to a numeric value in order to modulate access level. Zero or more Valued Attributes can be set in conjunction with the delegation of a Role. The set of Valued Attributes in a namespace is disjoint from the set of Roles in that namespace.

Form:	`[Subject → Object`
	` with A.ValuedAttribute1 <Operator> =<Value>`
	` <and B.ValuedAttribute2<Operator>=<Value>>*] C`
	A, B, and C may be all the same entity, or all different entities, or any combination thereof. The ”with” clause specifies the first Valued Attribute in the delegation, after which ”and” clauses may specify additional Valued Attribute settings.

Examples:	`[User → Server.Account with Server.DiskSpace = 10] Server`
	`[Intern → Server.Account with CPU.Priority-=5] Manager`

Delegation of Assignment for Valued Attributes	These delegations give the Subject the right to set the Object Attribute in future delegations written by the Subject. While the Valued Attribute is not a Role, the right to set it is a Role, and therefore can be the Object of delegations.
Form:	`[Subject → Entity.ValuedAttribute<operator>=’] Issuer`
Example:	`[Server.root → CPU.Priority-=’] Server`
Credential Management	These delegation annotations provide mechanisms to discover credential chains and control credential lifetime.
Discovery Tags	The Discovery Tag provides information to assist in the location of credentials across a distributed system.
Form:	`[Subject <Discovery Tag> →` . ` Object <Discovery Tag>] Issuer < acting as Role, Discovery Tag>` More information on discovery tags is provided in the Infrastructure section of this paper, section 4
Expiration Date	A date after which the delegation is no longer valid.
Form:	`[Subject → Object <expiry: date>]`

Table 2: Extensions to the dRBAC delegation model

reporters accredited by the US (`US.reporter`) by five hours. This can be achieved via the introduction of the following delegations:

```
[Officer.intelligence → Camera.view with Camera.delay=0] Camera
[US.reporter → Camera.view with Camera.delay=5] Camera
```

Note the extended delegation syntax: an object role followed by the “=” operator has the effect of setting the Attribute’s value.

Consider now a slightly more complex problem: The camera’s video feed can have multiple *independent* parameters. For example:

`delay`: how many hours to delay the feed

`rez`: resolution metric from 0 (fuzzy) to 1 (sharp)

There may be needs to modulate these parameters independently. To achieve this, we allow a delegation to specify multiple objects:

```
[Officer.intelligence → Camera.view
              with Camera.rez=1
              and Camera.delay=0] Camera
```

Third party delegation applies to these roles as well. The following delegations provide low quality video service to members of the press corps:

```
[Camera.fullRights → Camera.view’] Camera
[Camera.fullRights → Camera.rez=’] Camera
[Camera.fullRights → Camera.delay=’] Camera
[NSA → Camera.fullRights] Camera
[PressOffice.reporter → Camera.view
              with Camera.rez=1
              and Camera.delay=0] NSA
```

Finally, we consider the challenge of accumulated role values. For example, members of a foreign press corps may be authorized to receive the same video feed, however the PressOffice should be allowed to increase a particular delegee’s delay.

To allow this, we introduce a mechanism to build a composite values from multiple delegations. For example, the following delegation permits the PressOffice to increase the delay value by 24 hours for unfavored reporters.

```
[PressOffice → Camera.delay+’] NSA
[PressOffice.unfavoredReporter → PressOffice.Reporter
              with Camera.delay+=24] PressOffice
```

The “+” operator in the first delegation gives NSA the right to delegate Camera.delay, and the right to specify that it’s value should be summed with the value specified.

It is important that no entity can issue a credential that specifies a higher level of access or quality of service than they themselves receive. This is achieved in dRBAC through a careful choice of operators and valid value settings. Any associative operator with a top and a bottom can be used. Supported operators include:

+ : add a positive quantity to the valued attribute. Higher values indicate a lower quality-of-service. Base value is 0.

*: multiply the attribute by a positive quantity between 0 and 1. This will bound the range of possible values between 0 and 1. Larger values indicate a higher quality-of-service. Base value is 1.

<= : collects the minimum of all values along the certificate chain. Default value is infinity.

>= : collects the maximum of all values along the certificate chain. Default value is 0.

Valued Attribute Delegation Types The right to further modulate Valued Attributes can be transferred with either Object or Third-Party Delegation. Here is a general example of their use with Object Delegation:

```
[ A.a → A.b with A.v*=0.5] A
[ A.b → A.c with A.v*=0.5] A
```

implies `A.a` has role `A.c` with Valued Attribute `A.v` set to 0.25 of the base value.

Here is an example using third-party delegation:

```
[B.a = → B.b with A.r*=.5] B
[B → A.r*=′] A
```

implies `B.a` has role `B.b` with `A.r` set to 0.5 of the base value.

3.2.2 Credential Management

Table 2 also shows the syntax for *Discovery Tags* and *Expiration Dates*. Discovery Tags are described in section 4.

Expiration Dates are straightforward: Delegations can include a hard expiration date, after which the credential will no longer be deemed valid. A more interesting and flexible mechanism for controlling credential lifetime is provided by dRBAC's *Credential Subscriptions*, which allow for continuous monitoring of established trust relationships. Credential Subscriptions will be discussed at length in section 4.

4 dRBAC: Supporting Infrastructure

4.1 Functional Requirements

Deploying dRBAC in distributed environments requires a robust infrastructure that will perform the following functions:

- *Distribution:* provide a means for issuers of new delegations to "publish" them, so that they can be discovered by others;
- *Discovery:* when presented with the question "does entity x have role y?", locate a set of valid delegations composing a chain from x to y, or report that no such chain exists;
- *Validation:* once a delegation chain has been identified, verify the signatures of each delegation in the chain, and check expiration dates;
- *Monitoring:* when delegations are revoked or become invalid for any reason, notify any users who may have authorized actions based on the validity of those delegations.

This functionality is provided by dRBAC's *repositories.* In a distributed implementation, each participating server runs a dRBAC repository where users may publish credentials, submit queries, and subscribe to the status of an existing proof using *Credential Subscriptions*.

Because most of the complexity arises from the need to discover credential chains and monitor existing trust relationships, this section focuses on our solutions to those problems. *Validation* is comparatively straightforward: As in SDSI/SPKI [7], object delegations simply require a validity check of the issuer's signature. In addition to this check, third party delegations require validation of the support chains(s) that authorize the issuer to assign the object role and attribute value modifiers. Attribute value modifier operations are associative and are aggregated along the primary chain.

For clarity of presentation, we first describe the functionality provided by a single dRBAC repository, then discusses the ways in which repositories on different servers interact in a distributed implementation.

4.2 Abstraction Provided by a dRBAC Service

We first examine the abstraction of a centralized implementation of dRBAC, where the entire set of system credentials is stored in one repository. Our distributed implementation, which extends the semantics of the centralized repository, will be described later in this section.

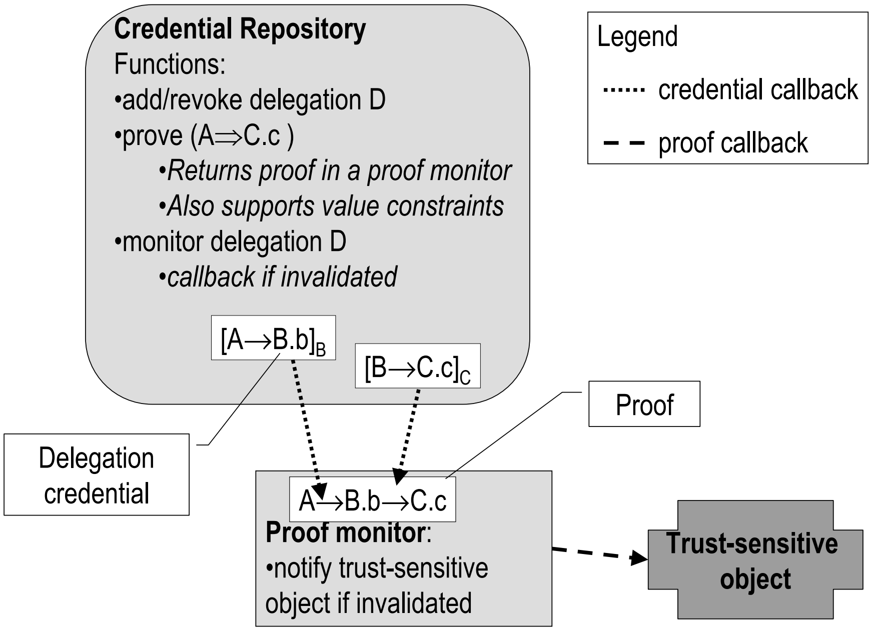

Figure 2: Centralized Implementation with dRBAC Semantics

The repository depicted in Figure 2 contains two delegations that supporting trust relationship between *A* and *C.c*. As indicated, a repository supports insertion and revocation of delegations, performs authorization queries that return proofs, and allows for continuous monitoring of discovered Proofs. Callback mechanisms are provided to inform Proof Monitors of credential status changes, and to inform trust-sensitive objects of proof invalidations.

Publication of new Credentials An Issuer of new Credentials posts these Credentials in the appropriate repository so they can be located and used by others. For efficiency reasons, the dRBAC implementation incorporates the design choice that a publisher of Third-Party delegations must provide the repository with all of the Support Chains needed to demonstrate the Issuer's right to delegate the Third-Party Role or Valued Attribute. This frees the repository from having to conduct recursive searches to collect the supporting chains when building proofs from sets of Delegations.

Queries against a repository A trust-sensitive object queries the repository in order to determine whether a requested access is permitted. The repository can be queried to return any proofs that demonstrate a trust relationship between the specified Subject `Sub` and Object `Obj`, and which satisfy a set of Valued Attribute constraints `C`. Our centralized repository implementation includes data structures that support efficient enumeration of credential chains between any specified subject and object.

In response to queries, repositories return Proofs. We will see that in the distributed implementation, a query may actually return portions, or *Sub-Proofs* of the desired proof. The returned proofs contain all necessary support chains to authorize third-party delegations.

Identifying a proof within the repository Internally, the repository builds a proof using a graph-based algorithm by submitting queries of the following two types:

- Object Queries: given a set of objects O and Valued Attribute constraints C, enumerate the full set of proofs whose Primary Chain terminates in O and that do not violate C.

- Subject queries: given a set of subjects S and Valued Attribute constraints C, enumerate the full set of proofs whose Primary Chain begins with S and that do not violate C.

Credential Subscriptions and Proof Monitors A fundamental objective of dRBAC is to implement support for continuous monitoring of trust relationships. For example, an army officer accessing a continually updated stream of classified radar imagery should have his access immediately terminated if he loses some necessary credential.

To achieve this, a dRBAC repository implements a pub/sub interface for each credential's validity. Using this, an agent monitoring the validity of a proof authorizing some trust relationship based on some set S of delegations is notified if a credential in S becomes invalid.

We have said that a query returns a proof (if one exists); in fact, what it returns is a proof wrapped in a *Proof Monitor* object. As described above, a Proof Monitor is responsible for registering *Credential Subscriptions* with a trusted repository for each of delegations in the proof. The Proof Monitor also includes a *Proof Callback* interface to the trust-sensitive object that first requested the proof to inform the trust-sensitive objects if the proof becomes invalid.

This design utilizes an event *push* data-flow model, which minimizes polling of a credential repository. When a credential in the proof is invalidated, the trust-sensitive object can either query the repository for an alternate authorizing proof or discontinue access. Similarly, if the repository initially cannot provide a proof satisfying the required relationship, the trust-sensitive object can register a callback that will be activated when such a proof is available.

4.3 Distributed Implementation

Distributed implementations of role-based trust management systems expose several problems not present in centralized systems. Credential discovery is a challenge since the full set of credentials required to authorize a trust relationship may be distributed over a set of hosts not previously known by any agent participating in a particular trust-sensitive interaction. Additionally, credential revocation requires that any copies of these credentials be reliably disabled, and that any dRBAC clients relying on their validity be quickly informed.

4.3.1 Credential Wallets and Coherence

The dRBAC infrastructure supports distributed repositories using a network of *Credential Wallets*. Individually, Credential Wallets implement abstractions provided by the centralized repository described above. Wallets include interfaces to communicate with other wallets in order to implement a cache of the delegations stored in a distributed repository. Typically, an entity specifies a *home* wallet where the *original* instance of a delegation lives. Wallets can also store copies of credentials whose home is in other wallets; in this case, these copies are termed *ghosts*.

Coherence of ghost copies is maintained using *Credential Subscriptions* from the credential's home or authorized proxies. An invalidation of an original credential instance results in notification of all wallets containing ghosts.

We now focus on how we can discover a credential chain that spans multiple wallets.

Discovery Tags To facilitate discovery across multiple repositories, our scheme annotates each potential delegation Subject, Object, and Issuer with a *Discovery Tag* that includes (1) a reference to an authorized

Home agent responsible for answering discovery and validity queries associated with it, (2) liveness monitoring constraints, and (3) search types that indicate the contexts in which credentials involving these entities and roles will be stored in various repositories. More precisely, discovery tags include:

- an Internet address and dRBAC role identifying (respectively) the role’s network-reachable authorized *Home* wallet monitor and a trust relationship required to authorize the home and its proxies.

  Example: `wallet.aol.com:AOL.wallet`

- A numerically valued *TTL* (time-to-live) field that indicates the number of seconds a credential is valid after validity confirmation from its Home wallet. A TTL value of zero indicates that a credential does not require monitoring.

- two discovery search flags, each of which can take on one of three values, indicate a home wallet that must contain a copy of credentials. Consider some delegation `D` giving some subject `Sub` the rights of object `Obj`.

  - There are three subject discovery types: ’`-`’, ’`s`’, and ’`S`’.: subject discovery types apply to the Subject of a delegation. If the subject of a delegation is of type ’`s`’ (store with subject) or ’`S`’ (search from subject), the delegation must be stored in the delegation’s Subject’s home. Type ’`S`’ also requires that all Object Roles that the subject can be granted must also be of type `S`.
  - There are three object discovery types: ’`-`’, ’`o`’, and ’`O`’. Like subject discovery types, object discovery types apply to the Object of a delegation, and the types have corresponding meanings. If the object of a delegation is of type ’`o`’ (store with object) or ’`O`’ (search from object), the delegation must be stored in the delegation’s Object’s home. Type ’`O`’ also requires that all Subject roles that are granted to the delegation’s Object must also be of type `O`.

The extended syntax for Roles and Entities with discovery tags is:

`RoleOrEntity(host:dispenseRole:disoveryFlags:TTL)`

For example, the following discovery tag indicates that the role `AOL.user` has a Home Wallet at `www.aol.com`, a credential distribution agent authorization role of `AOL.agent`, a TTL of thirty seconds, and has discovery types *searchable from subject* and *store with object*.

`AOL.user<wallet.aol.com:AOL.wallet:So:30>`

Although Issuers of third-party credentials are required to supply their repositories with all necessary support chains, it may become necessary at some point to discover new supporting delegations. This is also implemented using discovery tags. As potential subjects of support chains, issuers of third party delegations are annotated with discovery tags, and each third party delegation credential supporting remote discovery contains an additional *acting as* clause enumerating the assignment roles (including discovery tags) the issuer must be entitled in order to validate the credential.

```
[AOL.user<AOL.user tag> → APNet.user<APNet.user tag>
        with APNet.bw<APNet.bw tag> <= 10 ] APnet <APnet.tag>
        acting as <APnet.user′ tag, APNet.bw<=′ tag>
```

Discovery Algorithm Given these discovery tags, the credential discovery search process is straightforward. As in other distributed credential discovery systems, credential discovery requires support for directed searches from Subjects toward Objects and/or Objects toward Subjects.

The distributed discovery algorithm builds a complete proof by submitting Object Queries and Subject Queries to the wallets as directed by Discovery Tags. Our credential discovery mechanism extends the work of Clarke [6] and Howell [11] that support directed credential searches in both subject-toward-object and object-toward-subject directions by storing credentials in repositories that are referenced by their subjects and objects. A similar technique has also been recently investigated by Li and Winsborough [13].

Consider an agent seeking to discover a proof authorizing a trust relationship from Subject `Sub` to Object `Obj` satisfying a set of Valued Attribute constraints `C`. A credential discovery search must succeed if `Sub` has subject search type “S” or `Obj` has object search type “O”. Without loss of generality, the following description assumes that `Sub` (and therefore `Obj` has subject search type “S” (store with subject). The agent first queries its local wallet for the set of Sub-Proofs in which `Sub` is the Subject. If a proof is discovered, then no additional discovery is required.

Since `Sub` is of type “S”, all authorizing paths (should one exist) from `Sub` to `Obj` must consist only of delegations whose subjects are also of type “S” and therefore stored in the wallets associated with their subjects. For this reason, a subject-toward-object search will therefore discover a proof authorizing the requested relationship. The approach we utilize is a parallel breadth-first search. Due to the subject search type of “S”, the full set of delegations with `Sub` as subject can be obtained by querying `Sub's` home for the full set of proofs it can validate with `Sub` as Subject that also satisfy `C`.

The validated results of this query (or any subsequent query) can include a proof authorizing the required relationship. The search terminates when such a proof is discovered. Otherwise, the credentials returned from a query are inserted into the trusted wallet. The home wallets of Objects for which the local wallet supports a proof with S as Subject are queried for further delegations until a satisfying chain is discovered.

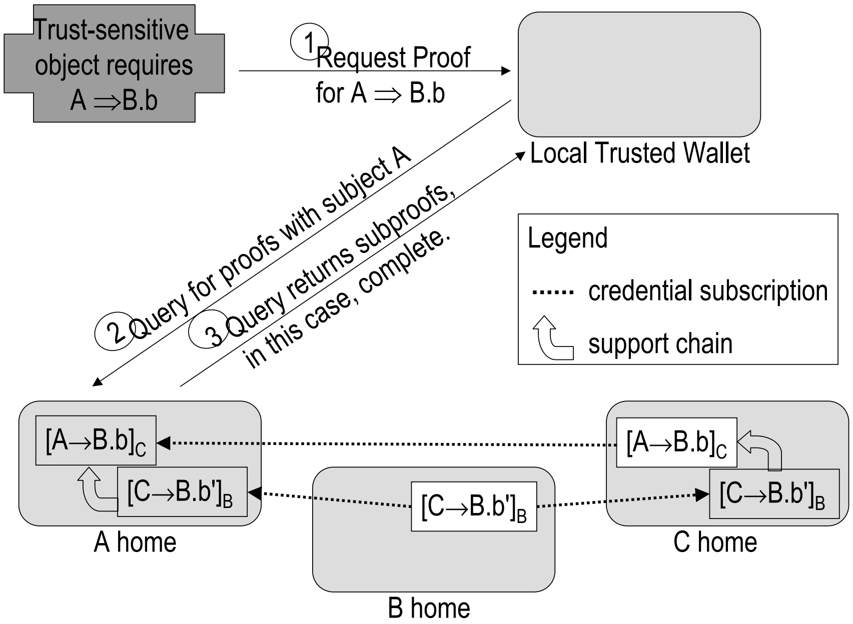
Trust-sensitive object requires A ⇒B.b
1 Request Proof for A ⇒ B.b
Local Trusted Wallet
2 Query for proofs with subject A
3 Query returns subproofs, in this case, complete.
Legend
credential subscription
support chain
[A→B.b]C
[C→B.b']B
A home
[C→B.b']B
B home
[A→B.b]C
[C→B.b']B
C home

Figure 3: Discovery: Initialization and first steps.

Figures 3 and 4 depict a subject-toward-object distributed search process. A trust-sensitive Object wants to authorize an action requiring Entity A to have the Role B.b. To do this, it requests that a proof demonstrating this relationship from its (trusted) local wallet. In this example, all Issuers, Subjects, Objects, and Assignment Roles are of type “search-from-subject”. Figure 3 portrays the wallets in their initial states and the first stages of the search.

Initially, the local wallet contains no delegations. Two delegations, [A→B.b]C and [C→B.b’]B, are stored in the distributed repository. Original copies are stored in the wallets corresponding to their Issuers’ homes, and Ghost copies reside in other wallets as required by their search types and support relationships.

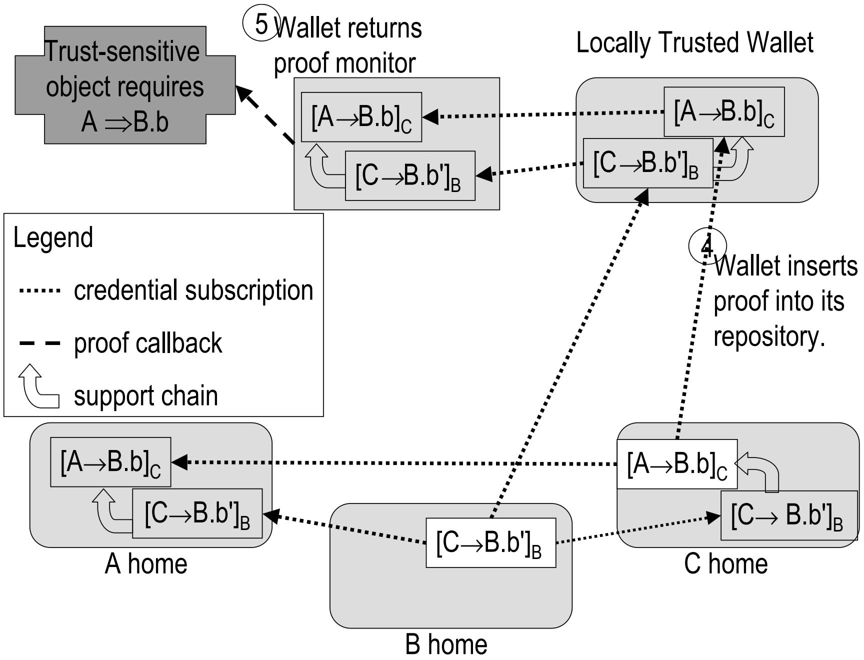

Figure 4: Discovery: Final Steps.

Dotted lines represent the inter-wallet credential subscriptions required for validation, and each step of the discovery process is labeled in the sequence they are executed; these steps are described below:

(1) The trust-sensitize object requests a proof-with-discovery from a trusted wallet. (2) Since the Requesting Wallet does not contain the necessary proof, A's Home Wallet is queried for all proofs in which A is the Subject. (3) This query's response contains a proof that satisfies the trust-sensitize object's requirements.

Figure 4 indicates the remaining steps in creating a proof monitor for the authorization: (4) Credentials from the proof are inserted into the local wallet *which is trusted to verify signatures and establish its own validation subscriptions*. (5) The local wallet discovers that the requested authorization is satisfied; a proof monitor is generated and is returned to the trust sensitive object.

Efficiency Considerations The number of potential authorizing paths in a tree with a constant branching factor is clearly exponential with depth. As observed by Winsborough et. al. [13], the number of paths to be considered may be significantly reduced if a credential search is simultaneously conducted in the *forward* (subject-toward-object) and *reverse* (object-toward-subject) directions. The dRBAC discovery algorithm similarly searches delegation paths in both directions, merges search results in the local wallet as additional credentials are collected, and terminates when a satisfying path is discovered.

Note that modulation of valued attributes may modify queries generated by a discovery process in a manner that effectively prunes the search space. For example, consider the problem of a search for a proof from some Subject E to Object H.h with constraint *E.x>0.05*. In a subject-toward-object search, if the only delegation discovered with F.f as Object is [E→F.f with E.x*=0.1]E, then the search from F.f should specify the modified constraint *E.x>0.5*. We note that this constraint will cause proofs containing the delegation *[F.f → G.g with E.x*=0.1]G* from being considered.

Multiple object queries with differing attribute value constraints may be required when multiple proofs are discovered from some Subject to a potentially useful intermediate Object I.i and multiple aggregate value modifiers. For example, consider a query for a path from H to J with value constraints $H.a > 0.02$

and $H.b > 0.02$. Assume two proofs P1 and P2 from S to I.i. P1 has has attribute value modifier $H.a* = 0.1$ and path P2 has attribute value modifier $H.b* = 0.1$. Clearly two families of potential proofs from I.i to the desired Object satisfy the required constraints. The first has constraints $H.a > 0.02$ and $H.b > 0.2$, the second has constraints $H.a > 0.2$ and $H.b > 0.02$.

Potential proofs are not necessarily discovered in topologically sorted order. Therefore, repeat queries may be required for multiple search paths that only modulate a single value attribute. Fortunately, all valued attribute modifiers are monotonic, so (depending on the order of proof discovery and the combination of valued attribute combinations) many of these repeat queries may be pruned.

Storage of Support Chains The dRBAC algorithm to extract proofs from a wallet can be extended to discover support chains for third-party delegations. However, we expect that prior to issuing a third party delegation the issuer will be aware of delegations that support their authorization to do so (and the ontology they represent). Discovery of support for arbitrary trust relationships (that might not even exist) is expensive; therefore dRBAC wallets do not automatically search for support chains. Instead the issuer is responsible for inserting third-party delegations complete-with-proof to all required wallets.

4.4 Implementation Status

dRBAC is being developed as a trust management component of a larger system called the Distributed Coalitions Infrastructure (DisCo) [9] that presents a simple unified interface for trust-monitored distributed application deployment. We have implemented a centralized dRBAC system that responds to trust-relationship queries generated by our DisCo infrastructure. Our current implementation is Java-based and uses Java RMI (Remote Method Invocation) and secure sockets implemented using our Switchboard [10] abstraction for inter-host communication. We are in the process of developing a distributed implementation of dRBAC. We intend to be in a position to distribute the full working version in a few months.

5 dRBAC in a larger security context: Case Study

dRBAC is one component of a larger architecture called the Distributed Coalitions Infrastructure (DisCo) [9]. DisCo presents a simple, unified interface for distributed application deployment in environments that contain systems and services administered by multiple authorities with changing trust relationships. DisCo provides application-neutral support for authentication and access control, secure communication, code distribution, and process rights management, thereby relieving the application developer from independently managing these features.

DisCo includes a dRBAC module to supply authentication and access control functionality. Application developers who are working with the DisCo infrastructure can make API calls to dRBAC to register new protected resources, or Roles. Thereafter, the DisCo infrastructure uses dRBAC to automatically handle the details of ensuring that all accesses to that resource are backed up by the necessary set of delegations, performing discovery to locate the full set of delegations, if needed.

We now develop an extended example of the type of problem that our DisCo infrastructure is designed to solve, highlighting dRBAC's contributions. We examine the formation of a *dynamic coalition* between a mobile user and the network facilities offered at an airport. The software that enables this transient trust relationship is enabled can be viewed as a single application consisting of multiple components:

- The communications client software the user runs on her laptop
- The gateway server software run by the airport network

Both of these components must support dRBAC as an authorization module.

Case Study Scenario:

BigISP and AirNet strike up a marketing partnership in which BigISP members can use AirNet's services in a limited fashion; i.e., with less bandwidth and server storage space, and fewer online hours per month. Sheila, who works in the marketing department at AirNet, administers the deal. Maria, a BigISP member, will attempt to log on to AirNet using her promotional membership.

Table 3 shows the delegations required to support this access. We now describe the process by which Maria's access is authorized by AirNet.

```
(1) [Maria → BigISP.member] BigISP
(2) [BigISP.member → AirNet.member with
             AirNet.BW≤100
             and AirNet.storage-=20MB
             and AirNet.monthlyHrs=10] Sheila
(3) [Sheila → AirNet.mktg] AirNet
(4) [AirNet.mktg → AirNet.member′ with
             AirNet.BW≤′
              and AirNet.storage-′
              and AirNet.monthlyHrs=′] AirNet
(5) [AirNet.member → AirNet.access with
             AirNet.BW = 200
             and AirNet.storage = 50
             and AirNet.monthlyHrs = 60] AirNet
```

Table 3: Delegations to support Maria's AirNet access.

Maria, a BigISP member, arrives at Springfield Intl. Airport. She attempts, for the first time, to use her laptop to log on to an AirNet server in order to connect to the Internet. She connects using a client that supports dRBAC as an authentication protocol; likewise, the AirNet service software that she is connecting to is enabled to use dRBAC.

Maria establishes a wireless connection to the AirNet server and requests permission to log on. She sends AirNet her public key and then proves to AirNet that she has the corresponding public key. Now the AirNet server must decide whether her access should be authorized. In other words, AirNet must discover a Proof that Maria$\Rightarrow$ AirNet.access.

AirNet knows that AirNet.access has the discovery search type "search from Subject." Therefore, it must contact Maria's Home Wallet and query it for all of the Proofs in which Maria is a subject. It discovers delegation (1), demonstrating that Maria is a BigISP member.

As instructed by the discovery tags in this delegation, the AirNet Wallet continues searching from the Subject side. It now contacts BigISP's Home Wallet, which returns a Proof consisting of delegations (2),(3), and (4) which grant AirNet.member to a BigISP.member. Delegation (2) is a *Third-Party delegation*, since the Issuing Entity is Sheila, and she is delegating roles and attributes defined not in her . Delegations (3) and (4) provides the support chain for this trust relationship.

Finally, AirNet queries its own Home Wallet for all Proofs in which AirNet.member is the subject. It discovers the self-certifying delegation(5). The *Valued Attributes* are aggregated, so that finally, Maria is granted AirNet.access with BW (bandwidth)$\leq$100, server storage of 30 MB, and a limit of 10 hours of monthly access.

Now that the proof consisting of these 5 delegations has been discovered, AirNet establishes *Credential Subscriptions* with Maria's Home Wallet for delegation(1), with BigISP's Home Wallet for delegations (2), (3), and (4), and with its own local repository for delegation (5). It delivers a Proof Monitor, containing a Proof Callback, to the server software controlling Maria's access, so that her access will be terminated in case any of these delegations should become invalid.

6 Related Work

SDSI/SPKI dRBAC builds on many of the features of SDSI [17]/SPKI [7]. SPKI/SDSI does not contain a notion of Third-Party Delegation, and therefore does not permit an entity to grant a role not mapped to its namespace. As described above, this restriction leads to the creation of names in multiple namespaces as *aliases* for each trust relationship. In SDSI/SPKI, certificate chains are linear in nature, rooted by the ACL belonging to application controlling access to the resource. The validation of SDSI chains depends upon the premise that the subject of certificate k is the issuer of certificate k+1. dRBAC extends this model to a graph-based approach in which Third-Party delegation allows the system to flexibly administer itself.

Credential Discovery Clarke et. [6] recognized the utility of reachability closures in credential discovery. We filter these closures for proofs that satisfy a required attribute value range restriction. Li and Winsborough [13] contemporaneously developed a credential discovery mechanism for their trust management system "RT0" that utilizes search tags with similar semantics to ours.

Credential Revocation dRBAC's online third-party certificate validity authentication scheme extends the mechanisms used by online positive authorization schemes such as OCSP [14] and revocation schemes such as CRLs [8] (The problem of efficiently broadcasting certificate revocation has spawned a surfeit of algorithms that efficiently encode and communicate sets of revoked credentials. For example, hierarchical schemes [1] and skip-lists [15]). OCSP provides an online mechanism to determine the validity of credentials. A client monitoring the status of an OCSP-certified must poll a authorized OCSP server, potentially creating hotspot agents communicating that no changes have occurred since the last time they were polled. Revocation-based schemes transmit information regarding all revoked certificates to all subscribers. Our subscription-based scheme permits the construction of hierarchical directory-based caches of trusted online validation agents that filter out revocation information irrelevant to particular caches. This scheme, which can be extended to utilize the aggressive encoding schemes utilized for traditional credential revocation schemes, minimizes network traffic for agents with long-standing trust relationships whose supported credentials are infrequently updated.

JXTA Sun's Project JXTA [16] has recently introduced a distributed credentialing system [5] that is designed to accumulate and distribute knowledge about the trustworthiness of peers in P2P applications. It does not have a notion of roles or delegations, and instead relies on a PGP-like ”web of trust,” in which you trust somebody who is trusted by others you trust, and modulation is achieved through the registering of ”personal opinions.” ”Web of trust” implementations generally do not provide a strong enough security model for use in security-critical applications.

PolicyMaker, Keynote, and Delegation Logic The entity whose namespace a role is defined and third party delegations in dRBAC provide similar power to the policy roots required for PolicyMaker [4], Keynote [3] and Delegation Logic [12]. All three of these systems all implement trust management systems that include another idiom for third-party delegations called "speaks-for" relationships. While Keynote and PolicyMaker support more complex trust relationships than dRBAC, they do not include mechanisms for delegation discovery or revocation monitoring.

Transitive trust can be limited by delegations that specify maximal depth is included in Delegation Logic. dRBAC can be extended to support such provisions; however we feel that the choice of depth limits is problematic and that other monitoring schemes may be more effective. In particular, the “S” “O” search tags that *require* public registry of further delegation provide an alternative mechanism to audit (and therefore restrict) re-delegation.

7 Conclusion

We have specified a decentralized access-control mechanism, dRBAC, for trust-relationships encompassing multiple administrative domains and discussed aspects of both its theory and its deployment architecture. We have introduced a method for Third-Party Delegation and shown how it augments the expressiveness of trust-management systems. Additionally, we have shown how trust-management languages can realize access-rights modulation through the use of Valued Attributes, eliminating the need for out-of-band security policy. Our supporting infrastructure introduces Credential Subscriptions, which solve the problem of efficiently monitoring established trust relationships for updates to credential status. dRBAC defines a complete system that can be used to distribute, locate, validate and revoke role-based delegations in a larger security context.

8 Acknowledgements

The semantics of third party delegations were refined with the assitance of Jordan Applebaum. Joshua Rosenblatt and Oliver Kennedy wrote proof evaluation and credential storage modules for early versions of the dRBAC centralized repository. Vladimir Vanyukov integrated public-key cryptography into dRBAC.

This research was sponsored by DARPA agreements F30602-99-1-0157, N66001-00-1-8920, and N66001-01-1-8929; by NSF grants CAREER:CCR-9876128 and CCR-9988176; and Microsoft. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA, Rome Labs, SPAWAR SYSCEN, or the U.S. Government.

References

[1] W. Aiello, S. Lodha, and R. Ostrovsky. Fast digital identity revocation. In *Advances in Cryptology - CRYPTO ’98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 1998.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System, Version 2. IETF Request for Comments 2704, Available at `http://www.ietf.org/rfc/rfc2704.txt`, 1999.

[3] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust management for public-key infrastructures. In *Proceedings of the 1998 Security Protocols International Workshop, Springer LNCS vol. 1550*, pages 59–63, 1998.

[4] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the IEEE Conference on Privacy and Security*, 1996.

[5] Rita Chen and William Yeager. Poblano: A Distributed Trust Model for Peer-to-Peer Networks. Sun Microsystems, Inc. White Paper, Available at `http://www.jxta.org/project/www/docs/trust.pdf`, 2001.

[6] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. Available at `citeseer.nj.nec.com/article/clarke99certificate.html`, 1999.

[7] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. SPKI Certificate Theory. IETF Request for Comments 2693, Available at `http://www.ietf.org/rfc/rfc2693.txt`, 1998.

[8] W. Ford. A public-key infrastructure for u.s. government unclassified but sensitive applications. Available at `citeseer.nj.nec.com/ford95public.html`, 1995.

[9] Eric Freudenthal, Edward Keenan, Tracy Pesin, Lawrence Port, and Vijay Karamcheti. DisCo: A Distribution Infrastructure for Securely Deploying Decomposable Services in Partially Trusted Environments (TR2001-820). Technical report, Department of Computer Science, New York University, 2001.

[10] Eric Freudenthal, Lawrence Port, Edward Keenan, Tracy Pesin, and Vijay Karamcheti. Credentialed Secure Communication Switchboards (TR2001-821). Technical report, Department of Computer Science, New York University, 2001.

[11] Jon Howell and David Kotz. End-to-end authorization. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2000.

[12] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. A practically implementable and tractable delegation logic. In *Proceedings of the 21st IEEE Symposium on Security and Privacy*, pages 27–42, 2000.

[13] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2001.

[14] Michael Myers, R. Ankney, A. Malpani, S. Galperin, and Carlisle Adams. Rfc2560: X.509 internet public key infrastructure online certicate status protocol. IETF Request for Comments 2560, Available at `http://www.ietf.org/rfc/rfc2560.txt`, 1996.

[15] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *Proceedings 7th USENIX Security Symposium (San Antonio, Texas)*, Jan 1998.

[16] Project JXTA. JXTA Version 1.0 Protocols Specification. Available at `http://spec.jxta.org`, 2001.

[17] Ronald L. Rivest and Butler Lampson. SDSI – A simple distributed security infrastructure. In *Proceedings of CRYPTO'96*, 1996.

[18] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control: A multidimensional view. In *10th Annual Computer Security Applications Conference*, pages 54–62, 1994.

[19] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 20(2):38–47, 1996.

[20] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Protocol Architecture. Internet Draft <draft-ietf-secsh-architecture-09.txt>, Available at http://www.ssh.com/tech, 2001.