dRBAC: Distributed Role-based Access Control for Dynamic Coalition Environments

Eric Freudenthal, Tracy Pesin, Lawrence Port,
Edward Keenan, and Vijay Karamcheti
Department of Computer Science
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 utilizes PKI identities to define trust domains, roles to define controlled activities, and role delegation across domains to represent permissions to these activities. The mapping of controlled actions to roles enables their namespaces to serve as policy roots.

dRBAC distinguishes itself from previous approaches by providing three features: (1) third-party delegation of roles from outside a domain's namespace, relying upon an explicit delegation of assignment; (2) modulation of transferred permissions using scalar valued attributes associated with roles; and (3) continuous monitoring of trust relationships over long-lived interactions. This paper describes the dRBAC model and its scalable implementation using a graph approach to credential discovery and validation.

1. Introduction

dRBAC was motivated by the problem of controlling access to resources in *coalition environments*. A "coalition environment" could be commercial, in which corporations form a partnership, or governmental/military, in which several nations work together to achieve a common goal. In either case, the defining characteristic is the presence of multiple organizations or entities that are unwilling to rely on a third party to administer trust relationships. Consequently, the entities must cooperate to share the subset of their protected resources necessary to the coalition, while protecting the resources that they don't want to share. The growth of mobile users and network-based services in the Internet promises to make such environments commonplace; however, supporting them presents a number of challenges:

- Highly dynamic coalition environments must support the authorization of resources at varying levels of access to reflect natural structures of organizations and their alliances. Additionally, these authorizations must allow transitive delegation to support access by users that are unknown to the resource owner.
- Established trust relationships must be monitored over their lifetime to track the status of revocable credentials. Continuous monitoring enables appropriate authorization of prolonged user-resource interactions such as login sessions or continuous data feeds.
- Credentials must be automatically distributed to those who require them. Entities must be able to discover credentials that authorize desired trust relationships.

Unfortunately, existing solutions do not address these challenges well. Access control lists are difficult to administer, and neither scale well nor permit transitive delegation of authority. Traditional role-based access control (RBAC) systems [16] depend upon a central trusted computing base administered by a single authority, which contains the entire organization's security policy. This approach does not scale to the large numbers of mutually anonymous users one might encounter in coalition settings. More recently, trust-based systems (e.g., SDSI/SPKI [15, 6], KeyNote [2], RT0 [11]) have been developed to control access in a decentralized fashion. dRBAC extends these models to include support for varying levels of access and continuous monitoring of trust relationships.

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 permissions for controlled actions in terms of *roles*. Roles are defined within the namespace of a responsible entity; these permissions can be transitively delegated to other roles within the same or other namespaces. 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 defined by public keys obviates the need to identify additional policy roots.

dRBAC has been influenced by other recently proposed trust management systems, most notably SDSI [15] and RT0 [11]. However, dRBAC distinguishes itself from previous approaches in its support for the following features:

- *Third-Party delegations* allow an authorized entity to delegate roles created by *another* entity by referring directly to the role originator's namespace. This mechanism, related to the *speaks for* relationship in the Taos system [17], improves expressiveness and permits a natural administration model where privileged entities in the system can create roles and designate other (less privileged) entities to give out these roles.
- *Valued attributes* allow for modulated control of access rights, supporting varying levels of access for the same resource. The issuer of a valued attribute delegation sets a numeric value associated with a role.
- *Continuous monitoring* allows dRBAC to guarantee validity of established trust relationships over the lifetime of prolonged interactions. This feature uses a pub/sub infrastructure to selectively push status updates for revocable credentials to interested parties.

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 a protected resource can define its own trust relationships with other entities throughout the distributed system. Privilege to access-restricted resources is completely specified through the issuance of authorization certificates called delegations.

Project Context dRBAC is part of a larger architecture called the Distributed Coalitions Infrastructure (DisCo) [7]. DisCo presents a simple, unified interface for application deployment in environments that contain systems and services administered by multiple authorities with dynamic and transitive trust relationships. DisCo provides application-neutral support for authentication and access control, secure communication, code distribution, and process rights management, relieving the application developer from their independent management.

DisCo utilizes dRBAC to manage authentication and access control. Application developers reference dRBAC to register new protected resources whose access is regulated using dRBAC roles. Thereafter, dRBAC facilities enable discovery of authorizing trust relationships between entities requesting interactions, and continuous monitoring of the status of these relationships over the interaction lifetime.

1.1. Extended Example

We now introduce an example that will run through this paper. In this "coalition", a mobile user seamlessly obtains Internet connectivity using network facilities at an airport by presenting evidence of a trust relationship between her regular ISP and the ISP administering the airport network.

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 access AirNet facilities.

Throughout the paper, we detail the ways in which dRBAC facilitates this partnership, including authenticating the principals to each other, specifying the usage restrictions placed on `AirNet` subscribers, helping `AirNet` servers discover evidence of the trust relationship between `BigISP` and `AirNet`, and efficiently monitoring `Maria`'s access once it has been established. By using dRBAC, `AirNet` will be able to grant `Maria` access at the appropriate level, *in spite* of the anonymous and transient nature of her relationship with the network.

The rest of this paper is organized as follows. Section 2 defines dRBAC terminology, and Section 3 discusses the basic dRBAC model for delegating authority and its extension to support modulation of access rights. Section 4 presents an architecture to support distribution, discovery, validation and continuous monitoring of delegation chains. We summarize the dRBAC facilities used in the above example in Section 5. We conclude with a discussion of related work and future implementation and design goals.

2. dRBAC Terminology

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

dRBAC uses the "building blocks" defined below:

Principals, resources, entities dRBAC does not distinguish between owners of *resources* protected by the system and *principals* attempting to access them. Both are termed *entities* and represented by a unique PKI public identity. Authenticating a principal in dRBAC consists of verifying (using standard public-key cryptographic protocols) that it does possess the claimed identity. Relaxing the distinction between resource owners and principals simplifies trust management without sacrificing expressiveness.

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. As in SDSI/SPKI [15] and RT0 [11], roles are names in an entity's namespace. dRBAC roles represent classes of permissions controlled by their namespace. These permissions can be delegated to other roles (in the same or other namespaces), or entities. For example, the role `BigISP.member` defines the name `member` in the namespace associated with the entity `BigISP`.

Delegations Delegations authorize a principal to act in a given role. At a high level, the format of a delegation is:

`[Subject → Object] Issuer`

where `Subject` is a role or an entity, `Object` is a role, and `Issuer` is an entity. The arrow "→" can be read as "has the permissions of." This relationship is cryptographically signed by the `Issuer`.

Transitive delegation chains Permissions can be delegated in a transitive fashion. A subject `S` who has been granted the permissions associated with a role `R` *may* be able to further delegate `R` to others, depending on how `S` was delegated those rights. (See Section 3). A sequence of delegations from a subject to an object is referred to as a *delegation chain*.

Proofs A graph of delegations that demonstrate that "principal `P` has the permissions of role `R`" is called a *proof* and authorizes `P` $\Rightarrow$ `R`. *Support proofs* validate the authority of an issuer to delegate a role.

Proof monitor In order to safely authorize prolonged trust relationships, dRBAC relies upon *proof monitor objects* that continuously monitor the validity of delegations comprising a proof. Proof monitors are described in Section 4.

3. dRBAC Delegation Forms

3.1. Base Model for Delegations

Table 1 gives syntax and usage examples for the base dRBAC delegation model, which includes three types of delegations: *self-certifying*, *third-party*, and *assignment*. The first two permit an entity to delegate permissions associated with a role, either in its own namespace or in another, while the third permits delegation of the "right of assignment" of the referenced role.

3.1.1. Self-certifying and Third-Party Delegation

In these two forms, the issuing entity delegates the permissions associated with a role to another role or entity. The general form of these delegations is as below:

`[Subject → OEntity.OName] Issuer`

where the delegation's `Subject` may be a role or an entity, its object `OEntity.OName` is a role in the namespace of `OEntity`, and `Issuer` is an entity. This delegation grants all privileges associated with `OEntity.OName` to entities with the privileges of `Subject`. If the subject is an entity, then these privileges may not be further delegated to others. Otherwise, if the subject is a role, these privileges can be extended to others via additional delegations.

Self-certifying and third-party delegations differ based on whether or not the object role belongs to the namespace of the Issuer. When it does, i.e., `OEntity = Issuer`, the delegation is referred to as *self-certifying* denoting that no additional authorization is required because an entity is permitted to delegate the permissions associated with any role in its namespace. All valid dRBAC proofs are rooted with self-certifying delegations.

When the object role being delegated is not defined in the namespace of the `Issuer`, the delegation is referred to as *third-party*. In this case, the `Issuer` must be delegated the right-of-assignment privilege for role `OEntity.Oname` as described below.

3.1.2. Assignment Delegation

dRBAC treats the "right of assignment" of permissions associated with a role, `R`, as if it were just another role itself, denoting it with a tick mark: `R'`. As with roles, rights-of-assignment can also be delegated, and the entity within whose namespace a role is defined can also delegate that role's right-of-assignment using self-certified delegations. Such delegations are referred to as *assignment* delegations and take the following form:

`[Subject → Object′] Issuer`

where `Subject` (which can be an arbitrary role or an entity) is given the *right of assignment* on the role `Object`. Such delegations authorize entities with the `Subject`'s privileges to directly delegate the `Object` role. Like other roles, right-of-assignment roles can be transitively delegated.

Assignment delegations authorize the third-party delegation form described above. To see this, note that using third-party delegation, an issuer can explicitly delegate a role that exists in another user's namespace. Each such delegation `D` in a dRBAC proof must be accompanied by a *support proof* providing the right-of-assignment of `D`'s object role to `D`'s issuer. Self-certifying assignment delegations represent the simplest form of a support proof. However, this idiom is recursive: a support proof may itself include third-party delegations, requiring additional support proofs.

Table 1 shows examples of self-certified, third-party, and assignment delegations. To see how these delegations come together to form a dRBAC proof, consider a situation where `Maria` wants to log onto `BigISP`'s servers by presenting delegation (3) (from Table 1), which identifies her as a

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:	Maria; BigISP
Roles	A name within an Entity's namespace.
Form:	Entity.LocalName
Example:	BigISP.member
Role Delegations	Signed Certificates that extend access rights on some Object to a Subject.
Form:	[Subject $\rightarrow$ 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:	
Self-certified Delegation	An Issuer A grants role A.a to some Subject. The role granted is defined within A's namespace.
Form:	[Subject $\rightarrow$ A.a] A
Example:	[Mark $\rightarrow$ BigISP.memberServices] BigISP .... **(1)**
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 $\rightarrow$ A.a']B
Example:	[BigISP.memberServices $\rightarrow$ BigISP.member'] BigISP .... **(2)**
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 $\rightarrow$ A.a]B
Example:	[Maria $\rightarrow$ BigISP.member] Mark .... **(3)**

Table 1. Syntax for the base dRBAC delegation model. Refer to the text for an explanation of the numbered delegations.

BigISP member.

Since the object entity (BigISP) and the issuer (Mark) are different, delegation (3) is of the third-party type, and so additional evidence is necessary to determine Mark's permission to delegate the role BigISP.member.

Delegations (1) and (2) in Table 1 provide this evidence. Delegation (1) grants entity Mark all permissions of the role BigISP.memberServices. Delegation (2) grants the role BigISP.memberServices the *right of assignment* on role BigISP.member, in effect allowing any entity that possesses permissions of the former role to also delegate the role BigISP.member to others. Both delegations (1) and (2) are self-certifying, and require no further authorization.

Delegations (1) and (2) compose a valid proof for Mark $\Rightarrow$ BigISP.member', which in turn acts as a support proof for delegation (3). Together, delegations (1), (2), and (3) prove that Maria $\Rightarrow$ BigISP.member, granting Maria the permissions of BigISP.member.

3.1.3. Benefits of Third-Party Delegation

Third-party and assignment delegations provide a semantically expressive structure with the following benefits:

Delegation clarity and namespace management It is natural for an entity E to directly express the delegation of a role (e.g., BigISP.member) by referring to that role's controlling namespace (BigISP). Without third-party delegation, E would require a distinct name in its namespace for each role that it wants to delegate (e.g., E.BigISPmember). The resulting namespace pollution significantly increases the difficulty of administering such systems.

Additional functional capabilities Third-party delegation provides a powerful mechanism for *grouping* assignment capabilities into a role R, which can be further delegated to users. Any entity with this "administrative" role now has the ability to delegate to others any or all of the privileges associated with R's roles.

This property, which we call *separability*, offers immense value in systems where a role may encompass a large number of capabilities. In its absence, the alternative is to create a number of local roles, one for each delegation scenario. The scalability implications (and resulting administration nightmare) of this expansion create incentives for administrators to create catch-all administrative roles that represent multiple privileges. Unfortunately, without separability, these aggregate roles can no longer be decomposed, resulting in a significant reduction in functionality.

3.2. Extensions to the Base Delegation Model

Table 2 gives syntax and examples for two extensions to the base dRBAC model – *valued attributes* associated with roles, and *credential management* information.

Valued Attributes	A name within an Entity's namespace, disjoint from the role 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.
Form:	`[Subject → Object with A.Attribute1 <Operator>=<Value> <and B.Attribute2 <Operator>=<Value>>*] C`
	A, B, and C can either be the same entity, or different entities, or any combination thereof. The "with" clause specifies the first Valued Attribute in the delegation; subsequent attributes are specified using "and" clauses.
Example:	`[BigISP.member → AirNet.member with AirNet.BW <= 100 and AirNet.storage -= 20] Sheila` .... **(4)**
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.Attribute <operator>=’] Issuer`
Example:	`[AirNet.mktg → AirNet.storage -= ’] AirNet` .... **(5)**
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.

3.2.1. Valued Attributes

Access to some services can be naturally differentiated using scalar values. For example, an ISP may provide differing bandwidth limits and guarantees based on individual customer agreements. To avoid an explosion in the number of roles, dRBAC allows association of scalar *valued attributes* with roles, which 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, albeit disjoint from the role namespace.

Zero 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 delegation's object, or that are inherited by that object. Finally, like roles, the right to delegate valued attributes can be assigned to third parties.

Valued attribute delegation types Valued attributes accumulate along delegation chains in a manner such that no entity is able to delegate greater permissions than they have themselves. This monotonicity of attribute values is guaranteed by restricting the range of scalar modifiers and associating each valued attribute with a single operator.

Supported operators include (see Table 2 for syntax):

$-$	subtract a positive quantity from the valued attribute. Default value is zero.
$*$	multiply the attribute value by a positive quantity between 0 and 1. Default value is 1.
$<$	collect the minimum of all values along the certificate chain. Default value is $+\infty$.

dRBAC also allows transfer of the right to further modulate valued attributes using either self-certified or third-party delegations.

To see an example of the use of valued attributes in the context of the extended example introduced in Section 1.1, consider delegations (4) and (5) shown in Table 2.

Delegation (4) accords the role `BigISP.member` all of the permissions of the role `AirNet.member`, but modulates them by defining how the scalar attributes associated with the latter role are modified. In particular, the bandwidth and storage limits of a `BigISP.member` role are defined relative to that of an `AirNet.member`: it receives a bandwidth of at most 100 units and 20 units less of storage. Delegation (5) shows an example of delegation of assignment for valued attributes: the role `AirNet.mktg` is given the right to modulate values of the `AirNet.storage` attribute.

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 have the obvious semantics for specifying credential lifetimes. dRBAC also provides an additional mechanism, *delegation subscriptions*, for updating credential lifetimes, which allow for the continuous monitoring of established trust relationships. Delegation subscriptions are described in the next section.

4. dRBAC: Supporting Infrastructure

The dRBAC infrastructure provides mechanisms for (1) publishing of delegations; (2) delegation discovery and validation to build proofs; and (3) continuous monitoring of credential validity. These mechanisms are implemented in distributed credential repositories, called *wallets*, hosted on participating servers. All user operations — delegation publishing, queries (of the form "Does principal P have the permissions associated with role R?"), and monitoring of existing proofs — are performed against a local wallet, which maintains a consistent view of the credentials in the system using *delegation subscriptions*.

Because most of the complexity arises from the need for delegation chain discovery and monitoring across distributed repositories, this section focuses on our solutions to those problems. For clarity of presentation, we first describe the functionality provided by a single dRBAC wallet, then discuss the ways in which wallets on different servers interact in a distributed environment.

4.1. Wallets

Similar to a real wallet containing identification cards, a dRBAC wallet stores a collection of delegations. Figure 1 shows the structure of a dRBAC wallet (containing two delegations that support a trust relationship between A and C.c) and its interactions with client applications. Wallets support the three basic operations described below:

Publication of delegations An issuer of new delegations posts these delegations in a wallet so they can be located and used by others. Issuers of third party delegations also must provide authorizing support proofs, freeing wallets from having to conduct recursive searches to collect the supporting chains when building proofs from sets of delegations.

Authorization queries A trust-sensitive system resource can query a wallet for proofs authorizing whether a requested access is permitted. The query specifies the subject that is requesting the access, the object being requested, and, optionally, a set of valued attribute constraints. dRBAC supports three kinds of queries:

- *Direct queries:* Given subject S and object O and valued attribute constraints C, does there exist a proof authorizing S $\Rightarrow$ O while satisfying C?.
- *Object queries:* Given an object O (more generally, a set of objects) and valued attribute constraints C, enumerate the full set of proofs that take the form * $\Rightarrow$ O and do not violate C.
- *Subject queries:* Given a subject S (more generally, a set of subjects) and valued attribute constraints C, enumerate the full set of proofs that take the form S $\Rightarrow$ * and do not violate C.

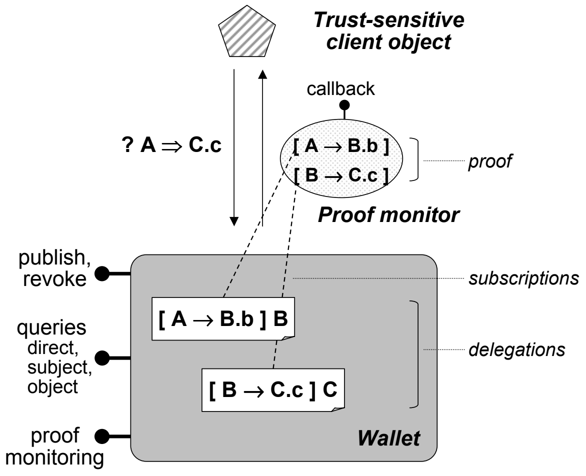


Figure 1. Structure of a single dRBAC wallet.

To generate these proofs, dRBAC wallets rely upon graph-based data structures that allow efficient enumeration of delegation chains between any specified subject and object. Generated proofs include necessary support proofs to authorize third-party delegations. The proof returned by a direct query, if one exists, suffices to establish a trust relationship between the subject and the object, thereby authorizing the requested access. Object and subject queries return portions of the overall proof, or *sub-proofs*, and assist in the construction of proofs spanning multiple wallets.

Proof monitoring Once a proof is returned, wallets provide functionality to allow continuous monitoring of its validity over the lifetime of the interaction. This is achieved via a callback mechanism involving an object called a *proof monitor* described in additional detail below.

4.2. Distributed Network of Wallets

A distributed environment poses two challenges. First, the full set of delegations required to authorize a trust relationship may be spread over multiple wallets. Second, revoking a delegation also requires revoking its cached copies and quickly informing any dRBAC clients that rely on its validity. The first challenge is addressed by augmenting credentials with *discovery tags* and using these tags during proof construction to discover required credentials, while the second is addressed by using *delegation subscriptions* to propagate delegation updates to all interested parties.

4.2.1. Discovery Tags for Delegation Chain Discovery

To facilitate discovery across multiple wallets, our scheme annotates each subject, object, and issuer of every delegation with a *discovery tag*.[1] The tag includes the following:

[1]dRBAC discovery tags have been influenced by the credential tag design proposed by Winsborough and Li in the context of RT0 [11]. The

- an Internet address identifying the entity's (or role's) authorized home wallet, e.g., `wallet.bigISP.com`.
- a dRBAC role required to authorize the home and its proxies, e.g., `bigISP.wallet`.
- a time-to-live (TTL) field that indicates the duration a delegation is valid following validity confirmation from its home wallet. Delegations that do not require monitoring are given TTL values of zero.
- two ternary discovery search flags, which specify the delegations one can expect to find (by design) in the entity's (or role's) home wallet.

  The *subject discovery* flag applies to the use of an entity (or role) as the subject of a delegation and can take three values: '`-`', '`s`', and '`S`'. Type '`s`' (*store* with subject) and type '`S`' (*search* from subject) require that such delegations must be stored in the entity's (or role's) home wallet. Type '`S`' additionally requires that all object roles that the subject can be granted must also be of type '`S`'.

  The *object discovery* flag applies to the use of a role as the object of a delegation and can take values '`-`', '`o`', and '`O`'. Similar to subject discovery types, types '`o`' (*store* with object) and '`O`' (*search* from object) require that the corresponding delegations be stored in the home wallet of the object role. Type '`O`' additionally requires that all subjects that this object role can be granted to must also be of type '`O`'.

An example of dRBAC discovery tags is shown below:
`bigISP.member`$<$`wallet.bigISP.com:`
`bigISP.wallet:30:So`$>$
which indicates that the role `bigISP.member` has a home wallet at `www.bigISP.com`, a distribution authorization role of `bigISP.agent`, a TTL of 30 seconds, and discovery types *searchable from subject* and *store with object*.

Although issuers of third-party delegations are required to supply their wallets 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 supporting remote discovery contains an additional *acting as* clause enumerating the assignment roles (including discovery tags) the issuer must be entitled to in order to validate the delegation.

Discovery algorithm dRBAC builds proofs requiring delegation discovery across multiple repositories by conducting searches from subjects towards objects and/or objects towards subjects (using subject and object queries against individual wallets) as directed by discovery tags. Our discovery mechanism extends the work of Clarke [5] and Howell [10]. A similar technique has also been recently investigated by Li and Winsborough [11].

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`. For our description, we assume, without loss of generality, that `Sub` has a subject discovery type '`S`'.

The agent first queries its local wallet for sub-proofs of the form `Sub` $\Rightarrow$ `*`, stopping if it finds one for `Sub` $\Rightarrow$ `Obj`. If not, the algorithm observes that since `Sub` is of type '`S`', all authorizing paths (should any exist) from `Sub` to `Obj` must consist only of delegations whose object roles are also of type '`S`'. This implies that any delegations where one of these roles appears as a subject must be stored in the wallet associated with the role. A subject-towards-object search will therefore discover a proof authorizing the requested relationship. Our algorithm utilizes a parallel breadth-first search, starting from a direct query for `Sub` $\Rightarrow$ `Obj` directed towards `Sub`'s home wallet.

If the query returns with a proof authorizing the required relationship while satisfying `C`, the search is terminated. If not, the algorithm issues a subject query for `Sub` to the same wallet. The returned proofs are inserted into the local trusted wallet, with the objects of these proofs serving as the roots for further searches.

A corresponding scheme, searching from object-towards-subject takes care of the case where `Obj` has object discovery type '`O`'. When `Sub` has discovery type '`s`' and/or `Obj` has discovery type '`o`', the algorithm starts by querying for the first set of sub-proofs from the corresponding wallets, and then based on the results, directs future searches.

4.2.2. Delegation Subscriptions for Monitoring Validity

A fundamental objective of dRBAC is to support continuous monitoring of trust relationships. To achieve this, a dRBAC wallet implements a monitored and secure pub/sub interface for each delegation (a mechanism to support such secure monitored links appears in [8]). These *delegation subscriptions*, implemented using an event *push* model to minimize polling, notify subscribers if the corresponding delegation is invalidated.

dRBAC uses delegation subscriptions in two ways:

Proof monitors 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. Proof monitors register delegation subscriptions with a trusted wallet for each delegation in the proof. Notification of changes in delegation status are communicated using a callback interface to the trust-sensitive entity that first requested the proof.

Upon receipt of this notification, the entity can request an

dRBAC scheme was developed independently but has been refined to incorporate relevant features from the RT0 scheme.

alternate proof or discontinue access. Similarly, if the wallet initially cannot provide a proof for the desired relationship, the entity object can register a callback that will be activated when such a proof is available.

Coherent caching of delegations Wallets can serve as validated caches for copies of delegations whose home is in other wallets. The copies are kept coherent by registering a delegation subscription with either the delegation's home wallet or an authorized proxy. Wallets containing cached copies are notified whenever a delegation is invalidated.

4.2.3. Efficiency Considerations

The number of potential authorizing paths in a delegation tree with a constant branching factor, as might be obtained by conducting a *forward* (subject-towards-object) or *reverse* (object-towards-subject) search, is clearly exponential in depth. As observed by others [11], a significant reduction in the number of paths that must be considered is possible if the search is simultaneously conducted in both directions, whenever allowed by the values of discovery tags.

Furthermore, monotonicity of valued-attibute values enables pruning of the search when the composition of valued attribute modifiers on its primary chain results in a value that does not satisfy the search requirements. Additional improvements are possible if wallet queries for path augmentation are accompanied by modulated attribute *ranges* that will satisfy the desired trust relationship in the context of constraints required by the chains already discovered.

Potential proofs are not necessarily discovered in topological order. Therefore, repeat queries may be required for multiple search paths that only modulate a single valued attribute. Monotonicity of valued-attribute operators guarantees that all searches will terminate and permits pruning of branches that do not satisfy attribute value constraints.

4.3. Implementation Status

We have implemented a centralized dRBAC system that responds to trust-relationship queries generated by our DisCo infrastructure, and are currently extending it to support a distributed network of wallets. Our Java-based implementation leverages a novel secure inter-host communication abstraction called Switchboard [8]. Updated versions of the dRBAC system can be downloaded from `http://www.cs.nyu.edu/pdsg/projects/drbac`.

5. A Case Study of dRBAC Usage

Returning to the example introduced in Section 1.1, we trace a full instance of dRBAC usage, including discovery, authorization, and monitoring. `BigISP` member `Maria` will take advantage of a coalition between `BigISP` and `AirNet` to obtain wireless Internet access through `AirNet`.

```
[Maria → BigISP.member] BigISP ................(1)
[BigISP.member → AirNet.member
    with AirNet.BW <= 100
    and AirNet.storage -= 20
    and AirNet.monthlyHrs *= 0.3] Sheila ....(2)
[Sheila → AirNet.mktg] AirNet .................(3)
[AirNet.mktg → AirNet.member′
    with AirNet.BW <=′
    and AirNet.storage -=′
    and AirNet.monthlyHrs *=′] AirNet .......(4)
[AirNet.member → AirNet.access
    with AirNet.BW = 200
    and AirNet.storage = 50
    and AirNet.monthlyHrs = 60] AirNet ......(5)
```

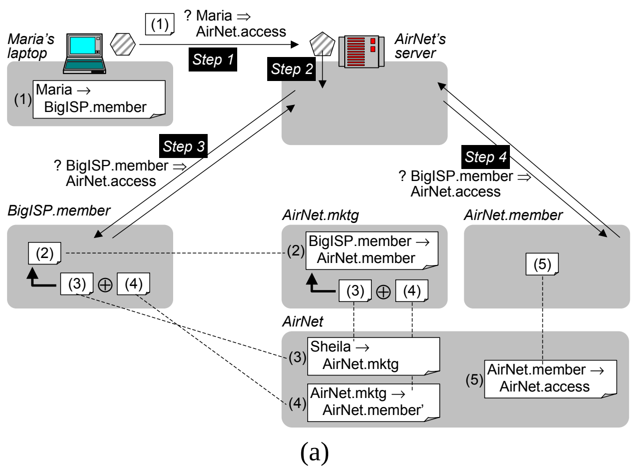
Table 3. Delegations supporting Maria's access to AirNet resources.

Table 3 shows the delegations authorizing this access. Delegation (1) identifies `Maria` as a `BigISP.member`. Delegation (2) defines the coalition between `BigISP` and `AirNet` as set up by `Sheila`, whose authorization for doing so is provided by delegations (3)–(5). All entities and roles in our example are assumed to be tagged with the subject discovery type '`S`' (search-from-subject).

Figure 2 shows how the case study evolves, from initial steps requesting authorization in Figure 2(a) to the final steps in Figure 2(b) that authorize and subsequently monitor the request. Figure 2(a) also shows the initial state of the various wallets: the server wallet is empty, and each delegation (along with its support proof) is stored in its subject's home wallet. Dotted lines represent the inter-wallet delegation subscriptions required for coherence.

Our case study starts off with `BigISP`'s software running on `Maria`'s laptop establishing a wireless connection to an `AirNet` server to connect to the Internet (*Step 1*). `BigISP`'s software authenticates itself to the `AirNet` server using a standard public-key cryptographic protocol, and requests access to the Internet on `Maria`'s behalf by passing on delegation (1), which validates `Maria` as a `BigISP.member`. To authorize access, the `AirNet` server software must discover a proof for `BigISP.member` ⇒ `AirNet.access`, which when combined with delegation (1) proves that `Maria` ⇒ `AirNet.access`.

Following the procedure outlined in Section 4.2.1, the `AirNet` server software queries its trusted local wallet for the required proof (*Step 2*). Upon failing to find a proof locally, the wallet attempts to discover the delegations necessary for it to build the proof. The wallet observes that the subject of the desired relationship, `BigISP.member`, has the discovery search type '`S`'. Therefore, it contacts the home wallet corresponding to the role `BigISP.member`


Maria's laptop
(1)
? Maria ⇒ AirNet.access
Step 1
Step 2
AirNet's server
(1) Maria → BigISP.member
Step 3
? BigISP.member ⇒ AirNet.access
Step 4
? BigISP.member ⇒ AirNet.access
BigISP.member
(2)
(3) ⊕ (4)
AirNet.mktg
(2) BigISP.member → AirNet.member
(3) ⊕ (4)
AirNet.member
(5)
AirNet
(3) Sheila → AirNet.mktg
(4) AirNet.mktg → AirNet.member'
(5) AirNet.member → AirNet.access
(a)

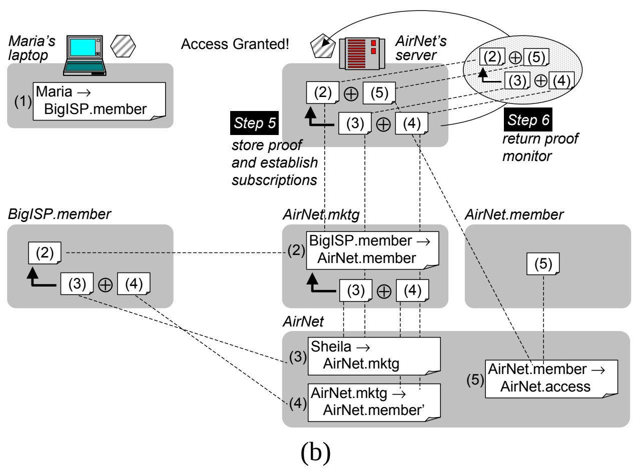


Figure 2. Distributed proof construction: (a) Initialization and first steps; (b) Final steps.

and issues a subject query to obtain all proofs of the form `BigISP.member` ⇒ `*` (*Step 3*). In response to this query, it discovers delegation (2), defining a relationship between the roles `BigISP.member` and `AirNet.member`. Note that since delegation (2) is a third-party delegation, the delegation must have been accompanied by its support proof. In this case, the latter comprises of delegations (3) and (4).

At this point, the server wallet has a chain from `Maria` to `AirNet.member`, but is still missing a proof that would authorize `AirNet.member` ⇒ `AirNet.access`. To obtain this, the wallet continues with its forward subject-towards-object search by contacting the home wallet corresponding to the role `AirNet.member`, and issuing to it a direct query for `AirNet.member` ⇒ `AirNet.access` (*Step 4*). The response to this query is self-certified delegation (5).

Now the proof authorizing `Maria` ⇒ `AirNet.access` is complete. Delegations from this proof are inserted into the local wallet, which is trusted to verify signatures and establish its own validation subscriptions (*Step 5*). The server wallet then aggregates the valued attributes, authorizing `Maria`'s access request with a BW (bandwidth) of 100 units (≤ 200), server storage of 30 units ($= 50 - 20$), and a limit of 18 hours ($= 60 * 0.3$) of monthly access.

The last step (*Step 6*), consists of the proof being returned to the original requester, the server software, bundled in a proof monitor object. As described earlier, this object allows the server software to continuously monitor the delegations authorizing `Maria`'s access.

6. Related Work

Decentralized access control Trust-management systems such as PolicyMaker [3], KeyNote [2], and Taos [17] permit expression of complex distributed trust relationships. These systems can in principle be used to support distributed access control, but need to be extended with credential discovery and revocation mechanisms.

dRBAC provides these latter features, and alternate techniques for some others. In particular, the mapping of roles to authorized name spaces provides expressive power similar to the policy roots required in these systems. Third-party delegations generalize the "speaks-for" relationship in Taos. On the other hand, unlike these systems, dRBAC does not currently support any provision for limiting transitive trust. While dRBAC can be extended to limit delegation depth, a scheme, which leverages '`S`' and '`O`' discovery tags to *require* public registry of further delegation may provide an alternative mechanism to audit and restrict re-delegation.

dRBAC is closely related to systems like RT0 [11] and SDSI [15]/SPKI [6] that combine role-based access control and trust management. dRBAC differs in its support for valued attributes and credential monitoring. Also, third-party delegations provide a cleaner mechanism for achieving role separability. In both SDSI/SPKI and RT0, the only way to allow a third party `T` to delegate a privilege `P` controlled by entity `O` is to introduce a phantom role representing `P` into `T`'s namespace. The resulting namespace pollution, along with the potential of accidental aliasing multiple privileges from multiple authorities to the same phantom role, complicates administration.

Recently, there has been a lot of interest in security models for peer-to-peer applications. Most proposed solutions, e.g., the credentialing system [4] in Project JXTA [14], rely on a PGP-like "web of trust", achieving modulation through the registering of "personal opinions". Unlike dRBAC, these solutions generally do not provide a strong enough security model for use in security-critical applications.

Distributed credential discovery Clarke et al. [5] recognized the utility of reachability closures in credential discovery. dRBAC filters these closures for proofs that satisfy a required attribute value range restriction. Li and Winsborough [11] contemporaneously developed a credential discovery mechanism in the context of the RT0 system that utilizes search tags with similar semantics to ours.

Credential validation and revocation dRBAC's delegation subscriptions offer benefits over both online positive authorization schemes such as OCSP [12] and revocation schemes such as CRLs [9]. Unlike OCSP, where a client monitoring the status of a certificate must continuously poll an authorized server (even when the credential has not changed), delegation subscriptions only require server and network resources when a credential has been updated.

Revocation-based schemes transmit information regarding all revoked certificates to all subscribers. In contrast, delegation subscriptions permit construction of hierarchical directory-based caches of trusted online validation agents that can avoid communication of updates irrelevant to particular caches. This scheme can yield additional benefits by utilizing the aggressive encoding and digest schemes, such as hierarchical structures [1] and skip-lists [13], developed for traditional credential revocation schemes.

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 delegation 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.

Acknowledgements

The authors thank Jordan Applebaum, Joshua Rosenblatt, Oliver Kennedy, and Vladimir Vanyukov for their assistance with the design and implementation of dRBAC.

This research was sponsored by DARPA agreements F30602-99-1-0157, N66001-00-1-8920, and N66001-01-1-8929; and by NSF grants CAREER:CCR-9876128 and CCR-9988176. 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 *Proc. of CRYPTO'98*, 1998.

[2] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust management for public-key infrastructures. In *Proc. of Security Protocols International Workshop*, 1998.

[3] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. of IEEE Conf. on Privacy and Security*, 1996.

[4] R. Chen and W. Yeager. Poblano: A Distributed Trust Model for Peer-to-Peer Networks. Available at `http://www.jxta.org/project/www/docs/trust.pdf`, 2001.

[5] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate Chain Discovery in SPKI/SDSI. Available at `citeseer.nj.nec.com/article/clarke99certificate.html`, 1999.

[6] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. SPKI Certificate Theory. IETF RFC 2693, 1998.

[7] E. Freudenthal, E. Keenan, T. Pesin, L. Port, and V. Karamcheti. DisCo: A Distribution Infrastructure for Securely Deploying Decomposable Services in Partially Trusted Environments. Technical Report 2001-820, Computer Science, New York University, 2001.

[8] E. Freudenthal, L. Port, E. Keenan, T. Pesin, and V. Karamcheti. Credentialed Secure Communication Switchboards. In *Proc. of IEEE Workshop on Resource Sharing in Massively Distributed Systems*, 2002. To appear.

[9] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. IETF RFC 2459, 1999.

[10] J. Howell and D. Kotz. End-to-end authorization. In *Proc. of USENIX Symp. on Operating Systems Design and Implementation*, 2000.

[11] N. Li, W. Winsborough, and J. Mitchell. Distributed credential chain discovery in trust management. In *Proc. of ACM Conf. on Computer and Communications Security*, 2001.

[12] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certicate Status Protocol. IETF RFC 2560, 1996.

[13] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. of USENIX Security Symp.*, 1998.

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

[15] R. L. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure. In *Proc. of CRYPTO'96*, 1996.

[16] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 20(2):38–47, 1996.

[17] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos Operating System. *ACM Trans. on Computer Systems*, 12(1):3–32, 1994.