Administering Access Control in Dynamic Coalitions

Rakesh Bobba[1] – NCSA, University of Illinois, Urbana-Champaign, IL
Serban Gavrila – VDG Inc., Chevy Chase, MD
Virgil Gligor – University of Maryland, College Park, MD
Himanshu Khurana – NCSA, University of Illinois, Urbana-Champaign, IL
Radostina Koleva – University of Maryland, College Park, MD

ABSTRACT

Dynamic coalitions enable autonomous domains to achieve common objectives by sharing resources based on negotiated resource-sharing agreements. A major requirement for administering dynamic coalitions is the availability of a comprehensive set of access control tools. In this paper we discuss the design, implementation, evaluation, and demonstration of such tools. In particular, we have developed tools for negotiating resource-sharing agreements, access policy specification, access review, wholesale and selective distribution and revocation of privileges, and policy decision and enforcement.

Introduction

In various collaborative environments such as alliances for research and development, health care, airline route management, public emergency response, and military joint task forces, autonomous domains form *coalitions* to achieve common objectives by sharing resources (e.g., objects and applications). Coalition resources may be privately or jointly administered and resource sharing is achieved by the distribution of permissions for coalition resources to coalition members based on negotiated resource-sharing agreements, or *common access states*. These coalitions are dynamic in that member domains may leave or new domains may join after coalition establishment. The focus of this work is on providing tools for administering access control in such coalitions.

Consider the following example of a coalition. Initially, three DoD/Intelligence domains form a coalition to respond to a threat situation relating to national security (e.g., monitoring transport of nuclear material, activities of terrorist groups). Examples of such domains can be Defense Intelligence Agency (DIA), National Security Agency (NSA), Defense Threat Reduction Agency (DTRA) and Central Intelligence Agency (CIA). These three domains decide to share privately owned resources (e.g., NSA shares a database of intercepted communication and the CIA shares intelligence reports) as well as jointly administered resources (e.g., integrated intelligence data and operation reports). Jointly administered resources are typically critical to coalition objectives and remain with the coalition even after the departure of member domains [16]. Access to jointly administered resources is one of the benefits individual domains derive from their membership in the coalition.

To obtain assistance from local authorities for a specific threat the DoD/Intelligence domains invite a civilian domain, say the Threat/Emergency Response Division of County C, to join the coalition. Upon resolution of the specific threat, County C leaves the coalition. Such domain join and leave events should not require the breakdown and setup of coalitions from scratch. Instead, access control tools must support dynamic domain joins and leaves – *Dynamic Coalitions* (DC) – in a seamless manner with ease and efficiency.

In this example, all four domains are collaborating to achieve a *common objective* (i.e., addressing threats to national security) that is desired by all domains and achievable by none of them individually. To achieve the common objective, each domain has a set of resources that are of interest to the others and is willing to share some or all of its resources with some or all of the other members.

It is important to note that the domains may not be willing to share these resources in the absence of this common objective; e.g., the DoD domains may be competitors for military tasks and would compete based on their available resources. Resource sharing by a domain with other domains means that the domain (administrator) grants access privileges for the resource being shared to the other domains. The sharing relationships among these domains are defined by the *access state* of the coalition. An access state is defined by the permissions each member domain has to the shared resources of that coalition.

Thus negotiating a *common access state* (CAS) means obtaining the agreement of each domain to share some of its data and applications with the other domains of the coalition. In addition, a CAS[2] of a coalition may include some jointly owned resources created during coalition operations, and a common policy for accessing these resources must also be negotiated.

[1] Author names listed in alphabetical order.

[2] Appendix B lists acronyms used in this paper.

The negotiation result is not merely a union of the contributed resources necessary to achieve a coalition objective. Instead, the negotiated CAS must satisfy both resource and permission constraints. In this work we consider all constraints that arise in the Role-Based Access Control (RBAC) model [8, 9, 10, 2]. For example, when the DoD domains initially form a coalition they may agree on a obligation constraint that requires at least one user from each domain to have administrative privileges for jointly administered resources.

Typically, these constraints arise from coalition objectives, access policies that are either jointly or privately enforced by autonomous domains, and resource-access requirements of coalition applications. The specification of negotiation constraints is an important part of any access-policy specification and drives the negotiation process; e.g., it determines the number of negotiation rounds and the convergence to and commitment of CAS. Hence, it must be defined in a precise manner [17].

A typical negotiation for this example would begin with the three DoD domains joining the coalition, specifying negotiation constraints, contributing resources that they are willing to share, proposing desired CAS, voting on the proposals based on satisfaction of constraints and extra-technological preferences, and committing to an agreed proposal. This negotiation process would be repeated when the civilian domain joins and leaves the coalition. Since the number of objects being negotiated and the number of rounds of negotiation could be large even in this small example, the negotiation process would be time-consuming and error-prone if undertaken manually. (In large coalitions such as those compromising tens of domains sharing hundreds of resources [22] this point is further emphasized.) Consequently, tools that fully (or at least partially) *automate the negotiation process* are needed. Automated negotiation tools would enable domain administrators to easily compose proposals for a CAS, verify that these proposals satisfy negotiation constraints, and commit them to production systems once agreed upon. Furthermore, for large-sized coalitions tools that enable administrators to *visualize the CAS* and to administer coalition resources via a graphical interface are very helpful.

In addition, *access review* tools are needed that would enable individual domains to review their local access state and verify that the state is secure prior to negotiation; i.e., that the state satisfies all local domain access policies. Based on this review, domains would be able to contribute resources to the coalition and compose proposals for CAS. Furthermore, access review for coalition resources may be undertaken at any point in time after coalition setup to (1) view the CAS along desired lines (e.g., subject permissions for a specific application, application access to specific objects), and (2) verify satisfaction of policies and constraints (e.g., Separation-of-Duty (SOD) policies, obligation constraints).

Once the CAS has been negotiated, tools that provide *wholesale, selective distribution and revocation* of access privileges are needed. Tools for wholesale distribution/revocation are needed as privileges must be granted to/revoked from *all* users of joining/leaving domains. Granting/revoking privileges to individual users would place undue overhead on administrators and, more importantly, would make it difficult to support coalitions where duration of domain membership could be smaller than coalition setup times using today's deployed technologies (e.g., weeks or months). Tools for selective distribution/revocation are needed as the tools must selectively target users of specific domains. One cannot, for example, exclude a member domain simply by modifying CA trust relations because those trust relations may be needed to share resources with that domain as part of another coalition.

There exists a need for coalitions in the commercial world as well as in government settings. What follows are two examples of commercial coalitions, taken from [15, 17]. First, consider a genetics research firm that discovers a gene sequence associated with a disease and establishes a coalition with a pharmaceutical company, two research hospitals and a Food and Drug Administration review board (FDA board) to find a cure using the gene sequence. Each domain shares some of its local resources with the coalition partners to achieve the coalition objective; e.g., the genetics firm contributes its gene sequence database, the pharmaceutical company provides a drug composition tool, the hospitals support clinical trials and give access to their patient databases (while preserving patient privacy by withholding sensitive patient information such as name, social security number, etc.), and the FDA review board shares a database of safety regulations. Given the impact of finding a cure, the coalition decides to jointly administer an application for remote consultation and drug analysis. This remote consultation and drug analysis application relies on a jointly administered secure group communication service. As coalition operations proceed, member domains may leave and new domains may join the coalition.

Another example of a coalition would be airline companies that collaborate to share various types of airlines routes in order to expand their market coverage. Sharing an individual route of a certain type implies that the airline domain that owns the route grants access permissions required to execute the route applications (e.g., reservations, billing, advertising) for that route to users of a foreign airline domain. Domains share routes (private resources) and may choose to jointly administer an auditing application that ensures adherence to coalition policies; e.g., policies on multi-hop route pricing and frequent flyer mile programs. As the coalition proceeds airline partners may leave or join the coalition.

In this paper we describe a suite of access control tools that we have developed for supporting dynamic coalitions. The tools employ the RBAC model and have been implemented in the Windows 2000 Server environment using Java. We have extended[3] an existing RBAC tool, called Multi-Domain Role Control Center (MDRCC), to provide access policy negotiation (i.e., CAS negotiation), specification, and review. The extended tool, MDRCC (Multi Domain Role Control Center), has been integrated with the Windows Active Directory for instantiating a negotiated CAS and to provide a Policy Decision Point (PDP) for resource providers. MDRCC provides a policy visualization language and a Graphical User Interface (GUI) that enables domain administrators to have a common view of coalition operations and to administer those operations directly from the interface. MDRCC has been integrated with an Attribute Certificate Authority (ACA) to provide wholesale, selective distribution and revocation of privileges. We use the Windows IIS web server as an example of a resource provider that enforces access policies for shared web pages. We have conducted experiments with our tools to show that they scale to support reasonable size dynamic coalitions and provide results of these experiments. We demonstrated the tools at Joint Warrior Interoperability Demonstration (JWID) 2004 and obtained valuable feedback.

The rest of this paper is organized as follows: We discuss related work, present the architecture of our system and discuss the salient features on our access control tools, describe the implementation of the tools, present experimental results, discuss lessons learned, and then conclude.

Related Work

Access control in distributed systems, in general, and in DCs, in particular has been extensively studied in the past. However, to the best of our knowledge ours is the first complete set of tools for access control in DCs that provide (1) support for negotiating access policies among domains for coalition set up, (2) wholesale selective distribution and revocation of privileges, (3) access review capabilities, and (4) support for jointly administering coalition critical resources.

Access Control in Dynamic Coalitions

The Secure Virtual Enclaves (SVE) [23] infrastructure allows multiple organizations to share resources while retaining organizational autonomy over local resources. SVE provides tools for (1) access policy specification via RBAC and (2) policy decision and enforcement via Access Calculators and Interceptor/Enforcers respectively. The SVE infrastructure is implemented in Java. Java RMI is used for intra-enclave communication and Ensemble group communication via JavaGroups Interface is used for inter-enclave communication. Resources in SVE are distributed applications based on Java RMI, Microsoft DCOM, Sun's Java Web Server and Microsoft's IIS.

The Yalta system [24] enables secure tuplespaces distributed across administrative domains. A tuplespace is a content-addressable shared memory. Entities in Yalta share information through a shared tuplespace that is distributed across administrative domains. Yalta provides for policy specification via Java's policy objects, policy decision and enforcement via Java Authentication and Authorization Service(JAAS) interface, and privilege distribution and revocation via a threshold CA and a Certificate Revocation Notification (CRN) service respectively. Tuplespaces in Yalta are implemented using JavaSpaces. The threshold CA is implemented in Java using the Yalta JCE provider, which, in turn, is implemented using Jini and Java RMI over SSL.

dRBAC [7] is a decentralized trust management and access control mechanism. The supporting infrastructure is implemented in Java and enables resource sharing across administrative domains. The tools provide for policy specification via RBAC, and provide privilege distribution and revocation and policy enforcement via a distributed network of wallets.

However, none of the above work (SVE, Yalta and dRBAC) provide tools for negotiating access policies, wholesale and selective distribution and revocation of privileges, reviewing access policy and joint administration of resources.

Phillips, et al. [21], describe a security model and enforcement framework that controls access to APIs of software that operate in a distributed environment running middleware like JINI or CORBA. The framework provides tools for policy specification via RBAC, policy decision and enforcement and privilege distribution and revocation via Unified Security Resource (USR). USR is a set of middleware resources (JINI and CORBA) that manage all mandatory access control (MAC) and RBAC meta-data for users, user roles and resources. However, the framework uses password based authentication that might cause longer coalition set up times. This work provides tools for access review but does not provide tools for negotiating access policies, jointly administering resources or wholesale selective distribution and revocation of privileges.

TrustBuilder [25] addresses the problem of credential negotiation for trust establishment between clients and resource servers. This work assumes extra-technological resource sharing agreement between domains and deals only with credential negotiation. This would not scale well in large coalitions where users have to negotiate credentials on every access. In our work we facilitate multi-party negotiations between domains to come up with resource sharing

[3] The initial version of MDRCC, called RCC, provided RBAC policy administration for an individual domain and was implemented in Visual Basic.

agreements and policies for member domains to gain access to those resources.

Access Control in Distributed Systems

Though the following work doesn't deal with coalitions explicitly, it is relevant to our work. Herzberg *et al.* [11] presents a Trust Policy Language (TPL) that allows organizations to define policies that map users to roles based on credentials presented by the user and those automatically retrieved by the system from credential repositories. RT [19] is a family of Role-Based Trust Management languages for representing policies and credentials in distributed authorization. KeyNote [4] is a decentralized trust management system that provides policy specification via PolicyMaker, policy decision via Keynote policy interpreter that takes the application's policy and credentials presented in the request and outputs a decision. It de-couples the application logic from access policy and authorization logic. The RBAC model employed by RCC may have fewer features when compared with TPL, RT, KeyNote, and dRBAC. However, RCC has an implementation that provides support for easy constraint specification and verification and an intuitive GUI for policy specification, visualization and administration.

Architecture and Access Control Tools

In Figure 1, we outline the architecture for administration of access control in a dynamic coalition comprising of two domains – which can easily be extended to *n* domains. We assume that each autonomous domain has its own Identity Certificate Authority (CA) that distributes identity certificates to users registered in that domain, and a container of users, resources (objects, applications), and privileges. To participate in the coalition each domain (administrator) installs the Coalition Resource Management (CRM) toolkit that consists of MDRCC, an Attribute CA, and a Secure Group Communication (SGC) toolkit. MDRCC provides specification of access control policies in the RBAC model, partially automated negotiation of the CAS, a visual common view of coalition operations, and access review capability. Once the CAS is instantiated on the container of users and resources, MDRCC provides a PDP via the container and administration of access policies for the users and resources via an intuitive GUI. The Attribute CA distributes (and revokes) attribute certificates (ACs) to both local and foreign domain users authorizing access to local domain resources. To jointly administer coalition resources, member domains establish a Joint Attribute Authority (JAA) that consists of an MDRCC module and a Joint Attribute CA (JACA), that is, an Attribute CA whose private key is shared among the member domains in a threshold manner [5]. As an example of a resource provider, coalition domains also establish local domain and coalition web servers that manage local and jointly administered applications (e.g., App O). The SGC component enables domains to communicate securely; e.g., to establish a CAS.

Policy Specification

MDRCC uses the RBAC access control model for policy specification. We chose the RBAC model for two reasons. First, it simplifies administration of access control for large systems by separating the

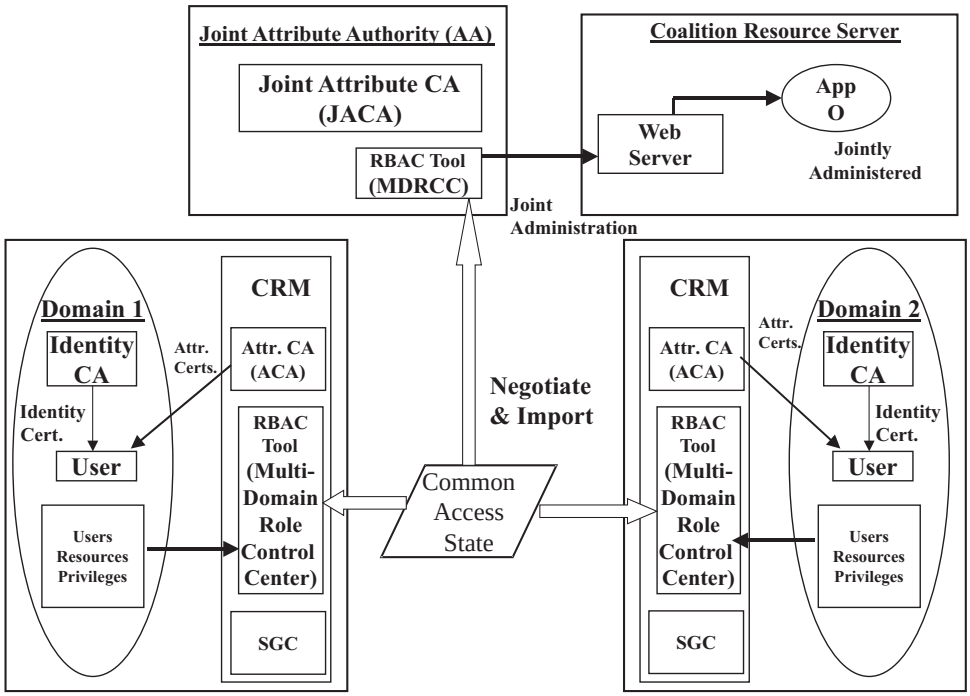

Figure 1: Architecture for access control in dynamic coalitions.

assignment of resource privileges to roles and users to roles [6]. Second, it supports easy specification and verification of constraints; e.g., Separation of Duty, and obligation constraints [8, 9, 2]. Though there are several policy specification tools based on RBAC (e.g., [7, 19, 11]) we chose MDRCC because it has been implemented, supports constraint specification and verification, and has a policy visualization language as well as an intuitive GUI that allows domain administrators to have a common view of coalition operations and to administer those operations from the GUI (e.g., add/remove users from roles).

Common Access State Negotiation

We use MDRCC and the SGC tool for partial automation of the process of negotiating a CAS among member domain. Administrators use MDRCC's GUI to encode negotiation constraints that can be either *global*, in which case they are known and agreed upon by all member domains, or *local*, in which case the constraints may remain private to some member domains. Types of constraints include Separation-of-Duty policies and cardinality constraints. Each domain administrator then uses MDRCC's intuitive GUI to specify access policies of resources they are willing to share and provides a view of these resources to other member domains. Any domain administrator then composes a proposal of a CAS (again using MDRCC's GUI) and sends the proposal to the other member domains. On receiving this proposal, domain administrators can view it and verify that it satisfies both global and local domain constraints (local constraints are also encoded using MDRCC's GUI). Domain administrators either register their vote on a given proposal or propose alternative CAS. Once all domains agree on a given proposal it is committed in that all domain administrators use MDRCC to instantiate the negotiated CAS on the local system; i.e., abstract objects and permissions in the CAS are mapped to actual objects and permissions of target systems.

Several approaches (e.g., agent-based) that fully automate multi-party access policy negotiations have been proposed (e.g., [3]) but have not yet been fully implemented. Though we are exploring such approaches, for the current system we chose the partially automated approach with MDRCC because, in practice, domain administrators would need to get involved in the negotiation as they would be responsible for administering the coalition and MDRCC provides these administrators with an intuitive GUI for the negotiation and administration.

Policy Decision and Enforcement

When a resource provider gets an access request it needs to verify the request against a specified access policy. To do so, the resource provider sends a request to a PDP and obtains the *policy decision*. In our solution this PDP is provided by the container where MDRCC instantiates the negotiated CAS. (As we will see later, Windows Active Directory is the container in our implementation that allows resource providers to send Lightweight Directory Access Protocol(LDAP) queries for policy decisions). This approach is scalable and is employed by many large-scale distributed systems; e.g., the XACML specification for web services [20]. After the resource provider obtains the policy decision it must enforce it. In our system we use a web server as an example resource provider and after obtaining a policy decision from the PDP the web server enforces it by allowing or denying access to the requested web page.

Distribution and Revocation of Privileges

Once a CAS is negotiated and instantiated, access privileges sanctioned in the CAS need to be distributed and we use attribute certificates to do so. We assign role memberships in attribute certificates and use a Public Key Infrastructure (PKI) for distribution and revocation of attribute certificates. Coalition users send requests to domain Attribute CAs for attribute certificates that grant them privileges to access the domain's resources. These requests includes the users' identity certificates. Attribute CAs verify the validity of the identity certificate and send requests to the PDP to verify the user's certificate request; i.e., to verify that the user has been assigned to the requested role in the CAS. Based on the reply from the PDP, Attribute CAs issue the attribute certificates. When users send access requests to resource providers with their attribute certificates, the resource providers verify the validity of the attribute certificates and query the PDP to verify if the roles assigned in the attribute certificates have the necessary privileges to access the requested resources. To revoke privileges, Attribute CAs simply revoke the attribute certificates and resource providers will no longer be able to validate the attribute certificates accompanying the access requests.

In addition to sharing privately owned resources, coalition members also benefit from jointly administering certain resources. Examples of such resources include intelligence reports, purchase orders for equipment, and financial data. In some coalitions, jointly administered resources may include auditing applications that are used to ensure that all domains are adhering to predefined access policies. These resources are typically critical for the coalition objectives and, therefore, must remain with the coalition even after the departure of member domains. To ensure the continuity of access to jointly administered resources in the presence of coalition dynamics, member domains setup a JAA. Since joint administration is consensus-based it is important to ensure that no single domain should be able to unilaterally define and modify access policies of a jointly administered resources [16]. To achieve this, the JAA comprises a Joint Attribute CA whose private key is distributed among the member domains; i.e., it uses threshold cryptography with each domain maintaining a share of the private key and using that

share for generating distributed signatures on attribute certificates [26]. The process of distributing and revoking privileges for jointly administered resources is similar to that for private resources except that signatures on the certificates are computed in a distributed manner with all domains participating to ensure consensus.

MDRCC is integrated with the domain Attribute CAs in each domain and with the Joint Attribute CA at the JAA to provide wholesale, selective distribution and revocation of attribute certificates. Administrators grant privileges to all users of a joining domain by just instantiating the CAS via MDRCC. MDRCC updates the container that acts as a PDP (w.r.t. certificate issuance) for the Attribute CA. Users of the joining domain can then request the Attribute CA (of the domain administering the shared resource) for attribute certificates before their first access to the resource and use the issued attribute certificate to access the resource there after. Whenever a user is issued an attribute certificate the Attribute CA registers it with its local MDRCC. Domain administrators can revoke all privileges of all users of a leaving domain just as easily as revoking the privileges of an individual user − with just a few clicks. When an administrator revokes user privileges via MDRCC's GUI, a request is sent to the local Attribute CA to revoke the respective attribute certificates. The Attribute CA revokes the attribute certificates and publishes a CRL.

Access Review

MDRCC uses the capabilities of the underlying RBAC language and a user-friendly GUI for access review and visualization. Domain administrators can (1) perform an access review along the desired lines (e.g., per-subject review, per-object review), and (2) verify the satisfaction of policies and constraints (e.g., Separation-of-Duty policies, obligation constraints) using MDRCC's interface. Also, once a policy is specified in MDRCC using the interface, MDRCC continuously monitors all future administrative actions for compliance.

Component Design and Implementation

Our tools are primarily implemented in Java and have been tested in Windows 2000 server environment. In this section we describe the implementation of the main components of our tools and the interaction between those components. In order for all the above components to inter-operate in a coalition environment, inter-domain trust relations must be established. For example, for MDRCC to verify signatures on CAS before instantiating it, MDRCC needs to have public-keys/certificates of the administrators and also trust them. These trust relations are set up manually before coalition set up using Windows Certificate Trust Lists that are easily composed and shared among the various components.

Multi-Domain Role Control Center (MDRCC)

MDRCC is a Java implementation of a Role-Based Access Control (RBAC) model [1] extended with general role hierarchies across multiple domains, static separation of duty and cardinality constraints, and advanced access review facilities. In MDRCC, each user or role is owned by a particular domain and the roles owned by a domain form a hierarchy based on the specified role inheritance relation. In each domain, the base of the role hierarchy is a special role called the domain base role. Users of any domain can be assigned to a role in a given domain. Roles are assigned abstract permissions to abstract objects. In addition, MDRCC includes data structures for (1) mapping selected portions of a role hierarchy within a domain to user accounts and groups on the domain controller, and (2) mapping the authorization information at the enterprise level (that is in terms of abstract objects and abstract permissions) to actual objects and permissions on the resources resident in various heterogeneous systems in the format required by native access control structures.

MDRCC is a three-tiered component that is made up of the following tiers: a Presentation Layer, an Application Logic Layer, and a Data Layer. The presentation layer comprises the MDRCC client(s), the application layer comprises the MDRCC server and MDRCC agents (resident in various target systems), and the data layer comprises the data repository. An MDRCC client provides a GUI for displaying the multi-domain RBAC model graph (or role graph), capturing user (Administrator) actions and sending them to the (remote) MDRCC server. The MDRCC server receives user (administrative) commands from the client, and executes them by accordingly updating the data layer. The MDRCC server is also responsible for mapping selected sub-graphs of the role graph (called views) to user accounts and groups on heterogeneous hosts (called also target systems), and for mapping abstract objects and role permissions to actual objects and permissions structures (e.g., ACLs) on those hosts. For these tasks, MDRCC uses agent software running on each host to create/delete groups and user accounts, and set up ACLs, according to commands received from the MDRCC server. The data layer consists of a directory service, which stores, retrieves, and protects the actual multi-domain RBAC data; i.e., the domain information, the user and role sets, the various relations, the abstract objects, and the mappings between multi-domain RBAC data and target system data. In our system, the Windows 2000 Active Directory provides the directory service; i.e., it is the data layer. The communication between MDRCC client and server is via SSL and is implemented using JSSE packages.

An MDRCC client not only provides visual access policy representation (see Figure 2) by displaying the role graph but also provides a very intuitive and user

friendly GUI for administrative actions; e.g., creating roles, assigning permission to roles. For negotiating a CAS, domain administrators use an MDRCC client to build a role graph (a CAS proposal) from the known set of resources that domains are willing to share and export the role graph into a compact, machine readable access state file. Other domain administrators then import the proposed CAS file and visualize it. They can verify that the proposed CAS satisfies local permission-based constraints using MDRCC's constraint verification facilities, and then either vote upon it or propose a new CAS. MDRCC currently supports the specification and verification of Static Separation of Duty (SSD) and cardinality constraints [8].

Once a CAS has been agreed upon all the administrators sign it with their private keys; i.e., to enforce agreement. The signed CAS is then imported into MDRCC at each domain and also at JAA. The CAS is then instantiated; i.e., abstract objects and permissions in the CAS are mapped to actual objects and permissions of target systems after verifying the signatures on the CAS.

Figure 2 shows a typical view of MDRCC's GUI and the Attribute CA GUI. MDRCC represents users and roles of the CAS as ovals of different colors, each color corresponding to a different domain. Roles are represented with solid ovals and users with solid ovals enclosed in an outer oval. Here we see two users from county C and a user from domain1 assigned to role "Manager". By right-clicking on any oval a drop-down menu is displayed which allows different operations for manipulating the MDRCC graph, setting and viewing role and user permissions, revoking user and viewing Separation-of-Duty constraints. Further options are available from the menu bar at the top which allow for changing the domain view, so that any domain can have a view of the entire CAS.

For access review administrators can perform either a per-subject review (reviewing all the permissions a subject has to all objects) or a per-object review (reviewing all users who have access permissions to a given object) with just a few clicks. For example, by right clicking on a user node in the graph and clicking "view permissions" in the drop down menu performs a per-subject review. Figure 3 shows how MDRCC allows for easy per-subject review of privileges. By left-clicking on the oval of a particular user, MDRCC displays all of the user's memberships to roles across all domains of the coalition. By right-clicking on each role and selecting view permission we can see the resources the user gets access to through that particular role and also the operations the user is allowed to perform on the resource. In some cases there is a need to view all of the user's accesses to coalition resources directly without viewing his role memberships, and MDRCC allows for viewing those accesses by right-clicking on the user name and selecting view permissions.

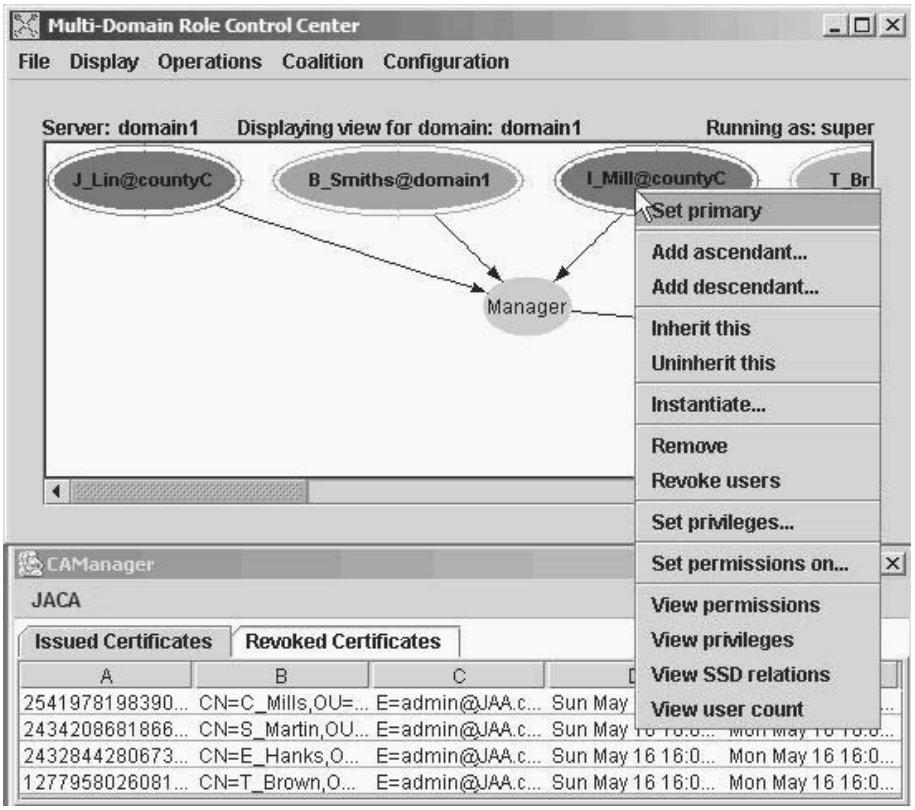

Figure 2: General view of MDRCC and Attribute CA during coalition operation.

Figure 4 shows MDRCC's interface for viewing and defining static Separation-of-Duty (SSD) permissions. The upper left quarter shows a list of defined constraints. A constraint is defined by a *role set* and a *threshold*. In this case we have one constraint with a role set called *ssd1* and a threshold of *two*. When a constraint is selected from the list, the roles that the role set of this constraint encompass are shown in the upper right quarter. The meaning of the constraint, *ssd1:2*, shown in the figure is the following: no more than two roles of the role set *ssd1* can have a common user. In Figure 3 we can see that this constraint is violated as user *J_Doe@domain3* belongs to more than two roles of the role set *ssd1* and MDRCC shows an appropriate error message and does not proceed further until the conflict is resolved. On the lower right quarter of Figure 4 are the coalition roles, for selection when creating new constraints.

Certification Authorities

Our system comprises three different kinds of CAs namely, domain Attribute CA, Joint Attribute CA and domain Identity CA. We assume that every coalition member has an Identity CA prior to coalition formation and hence do not deal with it here. However, for our testing and experiments we configured the CA service in windows 2000 server environment to act as an identity CA for member domains.

Attribute CA. Each coalition member domain has an Attribute CA that issues attribute certificates to both local domain and foreign domain users authorizing access to local domain resources. Attribute CA is a three-tiered component comprising the following tiers: a Presentation Layer, an Application Logic Layer, and a Data Layer. The presentation layer comprises the Attribute CA console (GUI), the application logic layer comprises the CA logic, and the data layer consists of key stores, issued certificates, published Certificate Revocation Lists (CRLs) and persistent data structures. The Attribute CA console provides an interface for administrative actions (e.g., starting the CA, stopping the CA, generating new keys, viewing currently issued certificates, viewing revoked certificates) and is implemented using *javax.Swing package*. The Application Logic Layer deals with issuing certificates, revoking certificates, and generating keys and is implemented using the Bouncy Castle Crypto API [13] and Java JCE packages.

Joint Attribute CA. Joint Attribute CA resides at JAA and issues attribute certificates to coalition users authorizing them access to jointly administered resources. Similar to Attribute CA, Joint Attribute CA is a three-tiered tool that comprises a Presentation Layer, an Application Logic Layer, and a Data Layer. But unlike Attribute CA, which uses a standard RSA cryptosystem, Joint Attribute CA uses a shared-RSA cryptosystem [5] and hence its Application Logic Layer and Data Layer are distributed among the coalition member domains. (In shared public-key cryptosystems the public key is owned by multiple principals with each principal having a share of the corresponding private key. The shared private key is generated in a distributed manner by all the participating domains (principals) and each domain retains a share

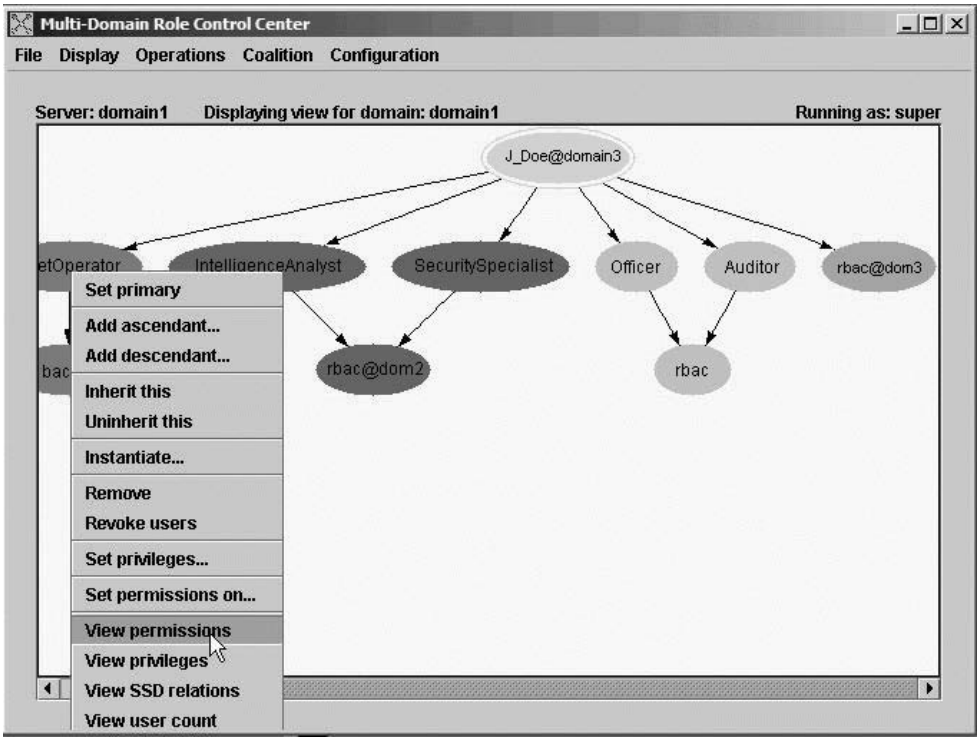

Figure 3: Per-subject review capability of MDRCC.

of the private key). These layers consist of (1) signature servers that reside in each domain for maintaining shares of the private key and signing certificates and CRLs and (2) a coordinator that resides at the JAA for coordinating the operations of the signature servers; i.e., the signature operation is distributed and is performed by the signature servers but is composed into a valid RSA signature by the Joint Attribute CA coordinator (see Appendix A for details on the employed shared-cryptosystem). The application logic layer is implemented using the Bouncy Castle Crypto API [13] and the Yalta JCE provider [24], which implements Boneh and Franklin's shared-RSA cryptosystem [26] with signature servers that communicate over Java RMI. We augmented the signature servers with a policy module (i.e., to enable policy checking capabilities). The Attribute CA/Joint Attribute CA GUI in Figure 2 displays the issued and revoked attribute certificates issued to both local users (in case of Attribute CA) and users from other domains. It also has menu options that allow the operation of Attribute CA to be suspended and started as needed.

Certificate issuance for Attribute CA and Joint Attribute CA is provided via a web server. Users authenticate to the web server using their domain identity certificates and upload their certificate request files, which are PEM encoded PKCS 10 files. Users then download issued PEM encoded X.509 V3 certificates. Both Attribute CA and Joint Attribute CA check with local MDRCC (which acts as a PDP for certificate issuance) and register issued and revoked certificates with MDRCC. In Joint Attribute CA the signature servers are also capable of checking certificate requests with local MDRCC to ensure that they comply with the CAS. Communication between the CAs (Attribute CA and Joint Attribute CA, both coordinator and signature servers) and MDRCC is via LDAP. When an administrator instantiates a negotiated CAS on the Active Directory, a PDP is automatically created that allows both Attribute CA and Joint Attribute CA to verify certificate requests providing wholesale distribution of privileges. If the instantiated CAS includes the departure of a domain then the administrator can revoke all users of that domain using the MDRCC client's GUI. This action results in the MDRCC Server sending a signed list of serial numbers (of certificates) that need to be revoked to the Attribute CA/Joint Attribute CA. Attribute CA/Joint Attribute CA process the request and publish a CRL on the web server, the URL for which is included in all certificates issued by the CAs. Attribute CA/Joint Attribute CA then register the revoked certificates with MDRCC via LDAP. Thus MDRCC and Attribute CA/Joint Attribute CA provide wholesale, selective distribution and revocation of privileges.

Figure 5 shows MDRCC's interface that allows for selective privilege revocation. Any domain (administrator) can selectively revoke the users from another domain (or all other) in a single operation. This figure in particular shows the MDRCC's ability for selective revocation where the privileges of users from another domain are revoked only for a specific role and not all roles in general.

Resource Server

In our implementation we use web pages as example resources and use the Windows IIS 5.0 web server as the resource server. To access web pages users submit attribute certificates to the web server, which checks the validity of the certificates by verifying the signature, checking the expiry time, and checking against the CRL for revocation information. After verifying the certificate (using the certificate verification functions built-in with IIS 5.0), the web server

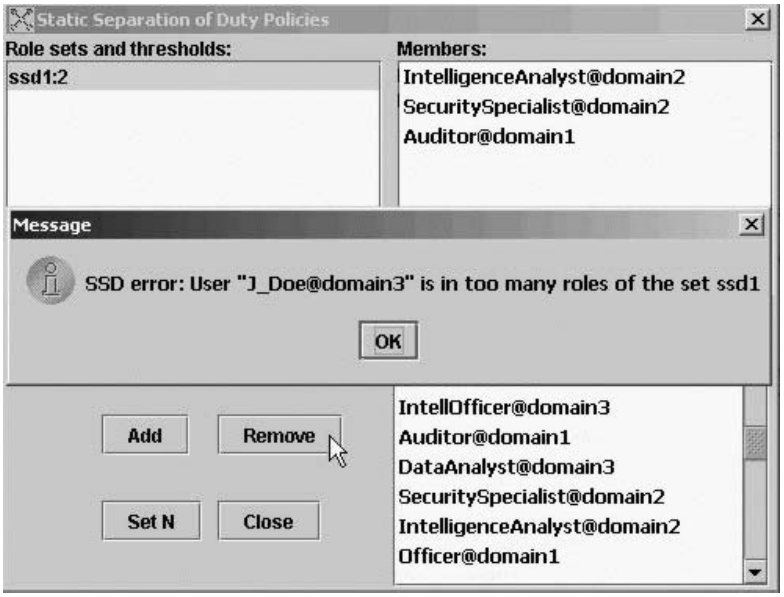

Figure 4: RCC's interface for specifying Separation-of-Duty constraints.

queries MDRCC (via LDAP) and grants or denies the request based on the authorization reply from MDRCC.

Secure Group Communication (SGC)

Secure group communication is needed in dynamic coalitions to aid administrators in negotiation during coalition set up and for smooth operation of the coalition after it is set up. Since the SGC requirement is not unique to dynamic coalitions we chose to use existing SGC tools. We use both Secure Spread [14] and Secure E-mail List Service [18]. The former provides secure, reliable multicast while the latter provides ease of use of e-mail in a secure manner.

Experimental Setup and Results

We set up experiments to measure the time taken for setting up a coalition of three domains and for domain join and leave events using our access control tools. Each domain in the experiment has ten applications that it shares with the coalition (private resources), ten roles that have permissions for the shared applications, and fifty users that get assigned to foreign domain roles and jointly administered roles for access to shared applications. Coalition members set up a JAA with ten jointly administered applications and ten roles with permissions for these applications. Coalition setup time is shown in Table 1 and includes time for importing a negotiated CAS (we do not include the time for negotiating a CAS as it is largely influenced by extra-technological decisions and actions), shared-key generation (since distributed shared-key generation is probabilistic the key generation times shown in the table are averaged generation times averaged over 10-15 runs), and certificate generation and distribution.

For coalition dynamics, we measured the time taken for a domain to join an existing coalition of three domains and for a domain to leave a coalition of four domains. The results are shown in Table 1. In case of domain departure we assume that the departing domain relinquishes its share of private key and hence there is no need to generate a new key. Note that this assumption only holds as long as the number of member domains that have left the coalition after the last key generation is not greater than *floor*(n/2) where n is

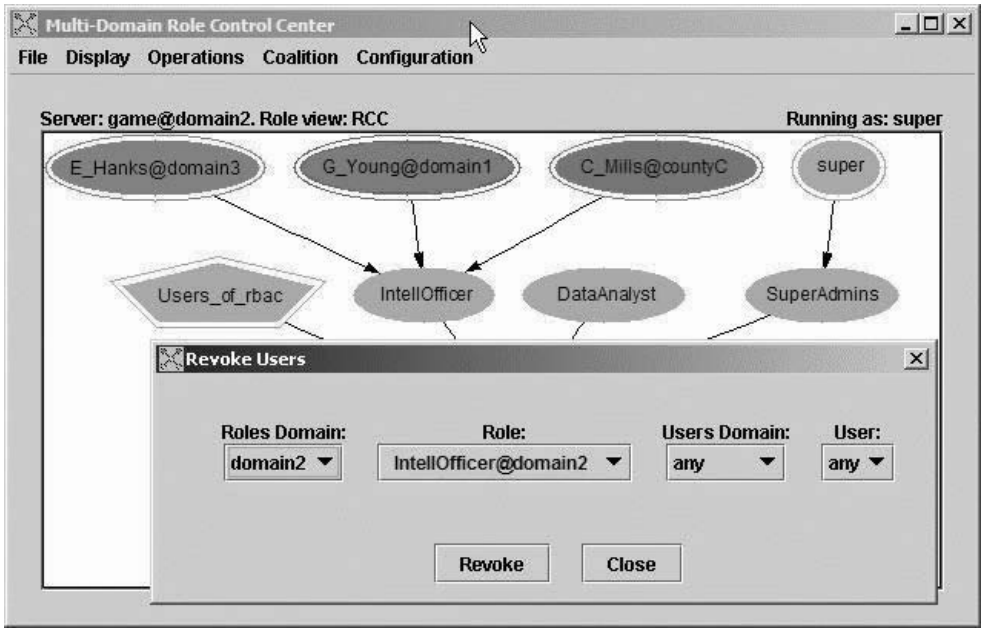

Figure 5: MDRCC's interface for wholesale, selective revocation.

	Shared-key generation (1024 bits)	Exporting CAS	Importing CAS	Cert. Dist. by JACA	Cert. Dist./ Revok. by domain ACAs	Total time
Coalition set-up with 3 domains	41 min	N/A	11 min	14 min (500 certs)	3 * 6 min (3 * 500 certs)	84 min
Domain Join	46 min	10 sec	16 min	17.5 min 600 certs	4 * 2 min (4 * 125 certs)	88 min
Domain Leave	N/A	10 sec	11 min	N/A	3 * 2 min (3 * 125 certs)	17 min

Table 1: Experiments for Coalition Setup, Domain Join and Domain Leave.

number of domains in the coalition at the time of key generation. Otherwise the member domains that left the coalition can collude to compromise the private key [5]. A domain leave event with key generation takes about the same amount of time as a domain join event. As can been seen from the measured times coalition set up and dynamics take only few hours at most for moderate sized coalitions there by illustrating the scalability of our access control tools.

Lessons Learned

The tools were demonstrated at various DARPA Principal Investigator (PI) meetings and most recently at the Joint Warrior Interoperability Demonstrations (JWID) in June 2004 [12]. JWID is an annual Chairman of the Joint Chiefs of Staff event that enables U. S. Combatant Commands and the international community to investigate command and control, communications and computer (C4) solutions that focus on selected core objectives. JWID 2004 assessed capabilities and technologies that allow information sharing with Homeland Security and Homeland Defense partners. This particular demonstration experience gave us a lot of useful feedback on our tools. JWID 2004 was conducted in a simulated operational environment. Our goal was to train users to use our tools who, in turn, demonstrated the usability of the tools to military personnel.

Our training experiences can be summarized in the following four lessons. First, we could train users (who had little security knowledge) in a very short amount of time to use our tools because 1) the tools enable visual representation of the CAS and provide an intuitive GUI and 2) we had generated and provided the users with extensive documentation in the form of task guides that take the users through execution of common coalition administrative operations step by step with the aid of images and animations. Second, the lack of security knowledge combined with multiple windows/interfaces of our tools intimidated the users. The users were expecting one unified tool (interface) but in reality we have a set of tools that integrate functionally and have individual interfaces. Third, in spite of our best efforts to test the system prior to the demonstration we still over looked flaws that were (inadvertently) detected in user training. For example, we did not prevent against multiple instantiations of MDRCC and users started multiple instances of MDRCC leaving the data layer in unknown states. Fourth, though our set of tools demonstrated the functional capabilities expected of them by the evaluators at JWID 2004, they were not as effective as we would have liked largely because users of the tools need to be knowledgeable of the underlying technologies in order to use them effectively. Our efforts in making them user friendly and providing good documentation helped us to a certain extent but did not eliminate the necessity to educate the users about the underlying technologies.

Conclusions

We have developed tools for administering access control in dynamic coalitions. In particular, tools for negotiating resource-sharing agreements, access policy specification, access review, wholesale and selective distribution and revocation of privileges, and policy decision and enforcement. Our experiments demonstrate that these tools scale to handle reasonable size coalitions. Demonstrations of the tools at JWID 2004 provided an opportunity for user training and obtaining useful feedback. Though we demonstrated our tools in Windows environment they can easily be ported to other platforms. In the future, we plan to implement the tools over web services for platform independence and rapid prototyping of enhanced features and capabilities.

Acknowledgements

We would like to thank Adam Moskowitz and the anonymous reviewers for their helpful comments and suggestions. Part of the first and fourth authors' work was funded by the Office of Naval Research under contract N00014-04-1-0562. The second, third and fifth authors' work and part of the first and fourth authors' work was funded by the Defense Advanced Research Projects Agency and managed by the U. S. Air Force Research Laboratory under contract F30602-00-2-0510. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, Defense Advanced Research Projects Agency, U. S. Air Force research Laboratory or the United States Government.

Bibliography

[1] ANSI INCITS 359-2004, *Information technology: Role based access control*, 2004.

[2] Ahn, Gail-Joon and Ravi Sandhu, ‘‘Role-based authorization constraints specification,’’ *ACM Transactions of Information System Security*, Vol. 3, Num. 4, pp. 207-226, 2000.

[3] Bharadwaj, Vijay G., and John S. Baras, ‘‘Towards automated negotiation of access control policies,’’ *POLICY ’03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, p. 111, IEEE Computer Society, Washington, DC, 2003.

[4] Blaze, Matt, Joan Feigenbaum, and Angelos D. Keromytis, ‘‘Keynote: Trust management for public-key infrastructures (position paper),’’ *Proceedings of the 6th International Workshop on Security Protocols*, pp. 59-63, Springer-Verlag, London, 1999.

[5] Boneh, Dan, and Matthew Franklin, ‘‘Efficient generation of shared RSA keys,’’ *Lecture Notes in Computer Science*, Vol. 1294, 1997.

[6] Ferraiolo, David, Janet Cugini, and Richard Kuhn, ''Role-based access control (rbac): Features and motivations,'' *Proceedings of 11th Annual Computer Security Application Conference*, pp. 241-248, Springer-Verlag, New Orleans, LA, 1995.

[7] Freudenthal, Eric, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti, ''drbac: Distributed role-based access control for dynamic coalition environments,'' *Proceedings of International Conference on Distributed Computing Systems*, pp. 411-420, Vienna, Austria, 2002.

[8] Gligor, Virgil, Serban Gavrila, and David Ferraiolo, ''On the formal definition of separation-of-duty policies and their composition,'' *RSP: 19th IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, 1998.

[9] Gligor, Virgil D. and Serban I. Gavrila, ''Application-oriented security policies and their composition (position paper),'' *Proceedings of the 6th International Workshop on Security Protocols*, pp. 67-74, Springer-Verlag, London, UK, 1999.

[10] Gligor, Virgil D., Himanshu Khurana, Radostina K. Koleva, Vijay G. Bharadwaj, and John S. Baras, ''On the negotiation of access control policies,'' *Revised Papers from the 9th International Workshop on Security Protocols*, pp. 188-201, Springer-Verlag, London, UK, 2002.

[11] Herzberg, Amir, Yosi Mass, Joris Michaeli, Yiftach Ravid, and Dalit Naor, ''Access control meets public key infrastructure, or: Assigning roles to strangers,'' *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, p. 2, IEEE Computer Society, Washington, DC, 2000.

[12] https://www.cwid.js.mil/public/cwid05fr/START_HERE.html .

[13] http://www.bouncycastle.org/ .

[14] http://www.cnds.jhu.edu/research/group/secure_spread/ .

[15] Khurana, Himanshu, Serban Gavrila, Rakesh Bobba, Radostina Koleva, Anuja Sonalker, Emilian Dinu, Virgil Gligor, and John Baras, ''Integrated security services for dynamic coalitions,'' An extended exposition abstract in *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III)*, 2003.

[16] Khurana, Himanshu, Virgil Gligor, and John Linn, ''Reasoning about joint administration of access policies for coalition resources,'' *ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, p. 429, IEEE Computer Society, Washington, DC, 2002.

[17] Khurana, Himanshu and Virgil D. Gligor, ''A model for access negotiations in dynamic coalitions,'' *WETICE '04: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'04)*, pp. 205-210, IEEE Computer Society, Washington, DC, 2004.

[18] Khurana, Himanshu, Adam Slagell, and Rafael Bonilla, ''Sels: a secure e-mail list service,'' *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, p. 306-313, ACM Press, New York, NY, 2005.

[19] Li, Ninghui, John C. Mitchell, and William H. Winsborough, ''Design of a role-based trust-management framework,'' *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, p. 114, IEEE Computer Society, Washington, DC, 2002.

[20] Moses, Tim, ''Core specification: extensible access control markup language (xacml) version 2.0,'' *OASIS Specification Document*, http://docs.oasis-open.org/xacml/access_control-xacml-2_0-core-spec-cd-04.pdf , 2004.

[21] Phillips, Charles E., Steven A. Demurjian, and T. C. Ting, ''Information sharing and security in dynamic coalitions,'' *Proceedings of IEEE Information Assurance Workshop*, 2002.

[22] Phillips, Charles E., T. C. Ting, and Steven A. Demurjian, ''Information sharing and security in dynamic coalitions,'' *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, p. 87-96, ACM Press, New York, NY, 2002.

[23] Shands, Deborah, Richard Yee, Jay Jacobs, and E. John Sebes, ''Secure virtual enclaves: Supporting coalition use of distributed application technologies,'' *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2000.

[24] Smith, T. J., G. T. Byrd, W. Xiaoyong, X. Hongjie, K. Thangavelu, R. Wang, and A. Shah, ''Yalta: A dynamic pki and secure tuplespaces for distributed coalitions,'' *Proceedings of 3rd DARPA Information Survivability Conference and Exposition*, Washington, DC, 2003.

[25] Winslett, Marianne, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu, ''Negotiating trust on the web,'' *IEEE Internet Computing*, Vol. 6, Num. 6, pp. 30-37, 2002.

[26] Wu, Thomas, Michael Malkin, and Dan Boneh, ''Building intrusion tolerant applications,'' *Proceedings of the 8th USENIX Security Symposium*, pp. 79-91, Washington, DC, 1999.

Appendix A: Shared RSA Public-Key Cryptosystem

In this section we discuss the use of shared public key techniques for generation and distribution of attribute certificates granting access to jointly owned resources.

Shared RSA Public Key Generation Algorithm

Here we review some of the features of the shared RSA public-key generation algorithm of [5].The algorithm enables n domains to generate a modulus $N = pq$ and exponents e and d. At the end of the computation all domains are convinced that N is the product of two primes, however none of them know the factorization of N. The public exponent e is made public while d is shared among the domains in a way that enables m-out-of-n threshold signature generation. That is, m domains are able to issue a certificate without reconstructing the key d. This also implies that an attacker who penetrates at most m-1 domains is unable to obtain any information about the private key. From the point of view of collusion the algorithm is (n-1)/2 private. That is, even if (n-1)/2 domains share the information they learn during the protocol, they will still not be able to recover the factorization of N or the private key d.

Joint Signatures with Distributed Private Key Shares

For joint administration of access policies, the public key K_{AA} of Joint Attribute CA (see Figure 1) is generated using the shared key generation algorithm resulting in private key shares that are distributed among all member domains (i.e., a n-of-n threshold sharing of the private key K_{AA}^{-1}). Once the public-key K_{AA} has been generated, all domains must apply a joint signature algorithm with their private key shares in order to sign attribute certificates with the private key K_{AA}^{-1}. The joint signature algorithm involves the Joint Attribute CA coordinator sending a message (the attribute certificate) to all the signature servers (co-signers) with the message M to be signed and a key ID comprising the hash of N and the public exponent e. Each of the co-signers then apply their corresponding private key shares d_i to compute $S_i = M^{d_i} \bmod N$ and send the computations back to the coordinator. The coordinator then computes the message signature $S = \prod_{i=1}^{n} S_i \bmod N$. This joint signature protocol is illustrated in [26]. Using this joint signature algorithm, the domains sign threshold attribute certificates (and CRLs) distributed by Joint Attribute CA. The joint signature algorithm also works for a threshold sharing of the private key; i.e., the private key K_{AA}^{-1} is shared in a t-of-n manner among the member domains. The algorithm is similar to the case above except that when the coordinator only needs computations from t servers and can compute the message signature $S = \prod_{i=1}^{t} S_i \bmod N$.

Appendix B: List of Acronyms

This appendix contains a list of acronyms used in the paper.

ACA Attribute Certificate Authority
ACL(s) Access Control List(s)
CA(s) Certificate Authority(ies)
CAS Common Access State
CRL(s) Certification Revocation List(s)
GUI Graphical User Interface
IIS Internet Information Service
JAA Joint Attribute Authority
JACA Joint Attribute Certificate Authority
LDAP Lightweight Directory Access Protocol
MDRCC Multi-Domain Role Control Center
PDP Policy Decision Point
PKI Public Key Infrastructure
RBAC Role-Based Access Control
SGC Secure Group Communication