A Framework for Role-Based Access Control in Group Communication Systems

Cristina Nita-Rotaru and Ninghui Li
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

In this paper we analyze the requirements access control mechanisms must fulfill in the context of group communication and define a framework for supporting fine-grained access control in client-server group communication systems. Our framework combines role-based access control mechanisms with environment parameters (time, IP address, etc.) to provide support for a wide range of applications with very different requirements. While the access control policy is defined by the application, its efficient enforcement is provided by the group communication system.

1 Introduction

Many collaborative applications such as phone and video conferencing, white-boards, distance-learning applications, games, shared instrument control, as well as command-and-control systems, have in common the need for a communication infrastructure that provides efficient message dissemination to multiple parties (often organized in groups based on a common interest), efficient synchronization mechanisms that allow for coordination and last, but not least, security services. Group communication systems (GCS) provide such services. Examples of group communication systems include: ISIS [9], Horus [21], Transis [4], Totem [6], RMP [28], Rampart [20], SecureRing [13], Ensemble[24] and Spread [8, 3].

An important aspect for secure collaborative groups is defining and enforcing a security policy. A set of definitions and requirements of security policies in groups is presented in [12]. The minimal set of security services that should be provided by any secure GCS and should be specified in a group policy include: client authentication, access control, group key management, data integrity and confidentiality.

While considerable research has been conducted to design scalable and fault-tolerant group key management protocols [29, 23, 5], and to provide data confidentiality and integrity [17, 2, 25, 7] for groups, less work focused on the access control services. When GCS are used as a common platform by several applications with different security requirements, there is an obvious need to control who can join a group, who can send/receive messages, etc. Major challenges when providing access control services to GCS are reconciling flexibility with scalability, and efficiently enforcing access control in the context of dynamic and distributed groups while supporting process failures and network partitions.

Most existing work in providing access control for groups employs traditional access control schemes such as Access Control Lists (ACL's). Such schemes make authorization decisions based on the identity of the requester. However, in decentralized or multi-centric environments, the resource owner and the requester are often unknown to one another, making access control based on identity ineffective or very expensive to maintain.

We adopt an approach in which the operations a client is allowed to perform depends on the role the client is playing in the group, and authenticated attributes of the client are used to determine which roles the client can play in a group. We focus on a GCS using a client-server architecture where the distributed protocols are run between a set of servers providing services to numerous clients. More specifically, our contributions are:

- We investigate the requirements for access control mechanisms in GCS and show why identity-based schemes do not provide enough flexibility to support a large class of collaborative applications.
- We design a fine-grained access control framework for GCS, based on ideas in Role-Based Access Control [26, 10] and RT [15], a Role-Based Trust-Management language. Our framework allows an

application to define its specific policies while the enforcement is performed in an efficient manner by the GCS. This is achieved by defining a set of basic group operations and roles that can be controlled and enforced by the GCS. Any application specific policy can be decomposed into these basic operations and application specific roles can be mapped to system roles.

- We analyze what are the implications of process (servers and clients) failures and network connectivity changes on the life cycle of a group policy in general, and of an access control policy in particular, and suggest how these issues can be addressed.

Roadmap We discuss the failure and trust models we use in Section 2. In Section 3 we present in details the components for a group policy, while in Section 4 we discuss the effects of process failures and network partitions on the life cycle of the policy. We overview related work in Section 5. Finally, we summarize our work and suggest future work directions in Section 6.

2 Trust and Failure Models

In this section, we discuss the trust and failure models we are using in this paper.

2.1 Trust Model

In client-server GCS, a trust model has to define the trust relationships within each layer (trust relationship between clients and trust relationship between servers) as well as between layers (i.e. do clients trust servers or not). Given this environment, several trust models are possible, ranging from a model where no entity trusts any other entity for any operation, both within a layer and between layers, to an optimistic model where servers and clients trust each other completely.

In this paper, we adopt the following trust model:

- Servers trust each other: In order for the system to be bootstrapped correctly, a list of legitimate servers should be provided to all servers, in the form of an ACL. Setting up this list is a system administrator's task and not an application task. We assume that there is a way to authenticate a server when it comes up and verify whether it is on the authorized configuration list. Once authenticated and authorized all servers trust each other. We note that in general the number of servers is small and that the way these systems are used is first define a servers' configuration that provides best performance for a specific network environment and application deployment. Therefore, in this case, an ACL is an acceptable solution.

- Clients trust servers to enforce the access control policy. This assumption is acceptable because, in the client-server GCS architecture, clients already trust the servers to maintain group membership and to transport, order and deliver group messages, so it seems natural to trust them also for enforcing the access control policy. Furthermore, this will allow for a more efficient enforcement since in numerous cases the decision can be made by each server locally, diminishing the communication overhead.

- Clients are not trusted (either by the other clients or by servers). Therefore, compromising one client does not compromise the security of the whole system.

2.2 Failure Model

Our model considers a *distributed system* that is composed of a group of servers executing on several computers and coordinating their actions by exchanging messages. The message exchange is conducted via asynchronous multicast and unicast. Messages can be lost or corrupted. We assume that message corruption is masked by a lower layer. A client obtains the group communication services by connecting to one of the servers. A client can connect locally or remotely. Both clients and servers may fail. When a server fails, all the clients that are connected to that server will stop receiving group communication services; they are not redirected to other servers.

Due to network events (e.g., congestion or outright failures) the network can be split into disconnected subnetwork fragments. At the group communication layer, this is referred to as a *partition*. A network partition splits the servers and can potentially split several client groups in different components. While processes (servers or clients) are in separate disconnected components they cannot exchange messages. When a network partition is repaired, the disconnected components merge into a larger connected component, this is referred at the group communication layer as a *merge*. First servers are merged, which in turn can trigger several client groups to be merged.

Byzantine (arbitrary) process failures are not considered in this work.

3 A Policy Model for Access Control in Group Communication Systems

In this section, we study the requirements for specifying access control policies in GCS and propose a policy model for doing so. Our goal is to design a policy model that is flexible enough such that it supports a diversified set of application policies, In addition, the policy model can be efficiently implemented by the GCS. The basic approach we use is as follows. For any group there is a set of basic operations that can be performed by principals (entities) based on their role, in a given context. *The mapping between group operations and roles, in a given context, defines the access control policy for that group.* This way, instead of having every individual application to implement and enforce its own access control mechanisms, we have applications defining specific policies that are translated to the set of basic operations that the GCS is aware of and can enforce access control on.

The rest of this section is organized as follows. We begin by describing an example scenario and discussing the various possible access control policies in Section 3.1. In Section 3.2, we describe the group operations that are subjected to access control. We analyze the use of roles in group policies in Section 3.3. We present the policy model in Section 3.4. In Section 3.5 we describe how a policy specified in the model is enforced. We discuss the challenges in maintaining the policy, while dealing with dynamic membership, failures and network partitions in Section 4.

3.1 An Example Scenario

Consider a virtual-classroom application implemented using a GCS. Multiple courses exist in the application. Each course has multiple sessions, each of which is represented by a virtual classroom, implemented as a group. For each course, there are instructors (some courses may have more than one instructors), TA's, and students. A classroom should be created only by an authorized user; thus a policy controlling the creation of groups must exist before the creation of a group. We call such a policy, a *template policy*. Each course has a template policy. Since template policies exist outside the context of any group and can be viewed as resources not specific to GCS, standard access control techniques are used to control the creation and modification of template policies. In the simplest case, only the GCS administrator is allowed to create or modify template policies.

A template policy determines, among other things, who can create a group based on the policy. One possible group creation rule is that only the instructors of a course are allowed to create a classroom for the course. An alternative rule is that a TA may also create a classroom. One may also allow the course instructor to delegate to another user, e.g., a guest lecturer, the authority to create a classroom.

After the classroom/group is created, a *group policy* needs to be created. A group policy can be created by copying the template policy. This group policy may then be tailored to suit the need of the current classroom session. Only authorized users should be allowed to change the group policy.

Various users may join the classroom in different roles, e.g., instructor, TA, student. Only authorized users should be allowed to join these roles. For joining as a student, different rules are desirable for different cases. Examples includes: only students who are enrolled in the class may join, the instructor or the TA's can admit additional students in special cases, or only students who are connecting from certain IP addresses may join (e.g., when taking an exam).

Several kinds of communication may be going on simultaneously in the classroom, and they should be subjected to different access control rules. For example, communication can be public: lectures delivered by the instructor, public questions asked by a student and the answers to those questions by the instructor or another member of the classroom. Some classrooms may allow any student to freely ask questions, other classrooms may require approval of the instructor before a student asks a question publicly. Communication can also be private, for example students may be allowed to ask questions privately to the TA's, or submit their answers to a quiz given in class. The instructor may be also allowed to eject a student from the classroom.

We note that most of the above services are provided by a GCS, without any access control enforcement. For example, the Spread [8] group communication system allows for multicast (public) and unicast (private) communication within a group, it also allows for any member to be both a sender and a receiver and can distinguish between different type of messages, while providing different reliability and ordering communication services. In addition, confidentiality and integrity of the data is provided.

3.2 Operations in Groups

From the above scenario description, we can extract the sensitive operations that need access control. The following operations are not performed within the context of a group, they precede the group creation and

are not subjected to a group policy or a template policy: 1) *create a group template policy* and 2) *modify a group template policy*.

A comprehensive list of basic operation that apply to a group and are the object of access control is presented below:

1. *create a group.*
2. *modify a group policy.*
3. *join a group.*
4. *send a message of a given type.*
5. *receive a message of a given type.*
6. *eject a user from a group.*
7. *destroy a group.*

The above list does not include the operation of leaving a group because this is an operation that can not be controlled. It is impossible to prevent a client from leaving a group [1].

We allow separate control for joining a group, sending a message, and receiving a message to provide support for a wide range of applications. For some applications several group members may be allowed to send, but not to receive messages. An example of such an application is a information reporting military application where clients use wireless communication; it is desirable to limit the information clients receive and store to minimize the damage caused in case of compromise. For other applications, some group members may be allowed to receive but not to send messages. For example, in a conference with a large number of participants only representatives may answer questions, while the rest of the participants are just listening.

3.3 Roles in Groups

One approach to specify and enforce access control is to use Access Control Lists (ACL's). Under this approach, a group has an ACL, which includes a set of users and the operations they are allowed to carry out. Such an approach is appropriate when the number of principals and operations is small and static. In general, ACL's have the following disadvantages. First, ACL's can get very large. For example, if every registered student in a university is allowed to join a classroom, then the ACL would be simply too long. Second, the ACL often duplicates information maintained in other places and its use in a dynamic distributed system will require maintaining its consistency across several sites which can be very difficult and prone to introduce inconsistency in the system.

From the scenario described in Section 3.1, it is clear that the set of operations a user is allowed to carry out depends upon the role that the user is playing in a group. For example, although a user may be the instructor of a course, in a guest lecture session she may be playing a TA or a student role.

We distinguish between two kinds of roles: *system roles* and *application roles*. System roles are predefined by the GCS; they exist in every group and have predefined meanings in terms of operations they are allowed to carry out. The following are system roles our framework supports:

- *(group) creator*: this role has at most one member, identifying the user that is the original creator of the group, i.e., the first member of the group. Because of failures, a group's creator role may be empty.
- *(group) controller*: this role has exactly one member, who has full control over a group, including changing the policy for the group and destroying a group. When a user creates a group, it is automatically made the creator and the controller of the group. We differentiate the group creator from the group controller for several reasons. First, the creator of a group may want to transfer the controller responsibilities to another member of the group; for example, a TA may create a classroom before the instructor comes and then, after the instructor joins, transfer the role to the instructor. Second, even when the group creator is the original controller, it may crash or leave the group, in which case another member needs to assume the group controller role.
- *(group) member*: any user who joins a group is automatically a member of this role.

Each system role comes with a set of allowed operations and has a set of operations that can be more fine grained defined. For example, for a client with the role *group member* restrictions on send and receive can be defined based on the message type.

Each group may also have a set of application-specific roles, for example, in the virtual classroom scenario, the following application roles may be needed: instructor, TA, student, auditor.

Once a user joins a group, the user may also perform the following operations related to roles:

1. *assume a role.*
2. *drop a role.*

[1] Any client can effectively leave a group by closing the connection with the server.

3. *appoint another user to a role.*
4. *remove another user from a role.*

We allow a client to drop a role at its will; however, the other three operations are subjected to access control.

The access control policy of the group defines the operations each role is allowed to carry out. In other words, a group access control policy maps each role to a set of operations. At any time, a user in a group plays a set of roles. When a user is about to perform an action, the roles that the user is playing are used to determine whether the action should be authorized or not. The roles and permissions that the application defines are mapped to system roles and operations a GCS is aware of and can enforce.

3.4 A Model for Access Control Policies in GCS

Clients must be authenticated before an access control policy is enforced. Several authentication mechanisms are commonly used. A GCS may provide a username/password based authentication mechanism or may use an external authentication system such as Kerberos [14, 18]. The client may connect with the server through TLS/SSL [1] with client authentication, in which case the client's public key and X.509 [22] Distinguished Name are available. Another solution is having the client to use certificates that document attributes of the clients, e.g., certificates in trust management systems such as RT.

The set of operations a client is allowed to carry out may depend on more than just the roles of the client; environmental factors may also have an effect. For example, a student may be allowed to attend a lecture if he/she is registered for the class and if the student joins the "class group" in a particular time frame, after the lecture started, he/she cannot join the group.

To accommodate the diversified authentication methods and the effect of environmental factors in access control, we introduce the notion of contexts. The GCS maintains a *client context* for each connected client and a *group context* for each group. A *group context* consists of a set of name/value pairs, in ways similar to Unix environmental variables. A group context provides environmental information such as current time and group state information (e.g., lecture has began in a classroom). The client context is similar to a group context; it stores information specific to a client, such as the IP address from which the client is connecting and the result of authentication (e.g., authenticated attributes of the client).

The combination of roles and context can accommodate a wide range of applications with very diverse policy requirements. A description of our model of group access control policies, as well as an example policy are presented in [19].

3.5 Enforcing Access Control in Group Communication Systems

When enforcing access control in GCS it is very important who is making the access control decision and who is enforcing it. Remember that we consider a client-server architecture, where service to clients (organized in groups) is provided by a set of servers. Many groups can exist in the system.

One solution is to have access control enforced by group members (clients). Although this approach seems appealing because in fact access control policies are group specific, it decreases the scalability of the system since each group must perform its own enforcement mechanism. Additionally, when access control is performed by clients, access restrictions such as dropping messages and requests at the receiver are more difficult to provide.

As clients are already trusting the servers for maintaining group membership and delivering and ordering correct information, the security model is not weakened by requiring the servers to also perform the access control enforcement, the potential benefit being increased scalability and more flexibility of the operations that can be enforced. Based on group's policy, servers must first reach a decision, if access is granted or not, and then enforce that decision. We distinguish between two general approaches:

- local decision: only one server is required to make a decision. For example, when a client requests access to a group during a join operation, the server the client is connected to can make the access control decision locally based on the client's role, group name and group policy and enforce it immediately.
- distributed (collaborative) decision: the policy requires several servers to collaborate in order to reach a decision, by using for example a voting mechanism, such as a given percentage of group members of a certain role have to approve. This approach requires a complete view of all the members of all roles of a group, information available to the servers.

One fundamental question is how does the application specific access control policy translates into a policy that the GCS understands. This translation can be

operated by a *Policy Translation Engine* that parses the group policy and outputs another file that the GCS will use in making/enforcing access control, file that defines permission based on the roles and operations that the GCS implements. Two additional operations are required once a policy is in place. The first one involves a check on making sure that the policy does not include any contradictory rules. The second one relates with the one the policy is distributed to the other servers and make sure that all servers have the same policy. In case the policy is static all is needed is that the policy is certified (digitally signed) and distributed by a server. In case the policy is dynamic, the policy file should be treated as replicated data among the set of servers.

Besides decision reaching, another important aspect is who is enforcing an operation. For most of the operations, the enforcement can be done locally by the server that makes the authorization decision. For other group operations, such as group destroying, the server enforcing the decision can be different from the one making the decision. For lack of space we could not include a detailed description on how enforcement is performed on each group operations. This information is available in [19].

4 Life Cycle of an Access Control Policy

In the previous section we described how a fine-grained access control policy for GCS can be defined and enforced in a model where faults do not happen. Unfortunately, this is not the case in the real world where processes can crash, computers can fail, network mis-configurations can happen, or the network overload can create unusual latencies that can be perceived as network partitions. In this section we examine how failures and network connectivity affect the life cycle of the policy.

The life cycle of a policy is defined by the policy creation and subsequent updates. As described in the previous section we assume that based on an application policy's specifications a group template is generated. The creation and revision of a group template is handled by the administrator of a GCS. Based on the template, a group policy is created when a client allowed to create groups, creates a group based on the template.

An access control policy can be static, in other words it can never change during the life of the group, or it can be dynamic, in which case it can suffer changes. In case of dynamic policies, a policy reconciliation must be performed in many cases. As shown in [16], policy reconciliation cannot always be solvable, in which case the question is what happens to the group. For example, current group members that do not satisfy the policy anymore can be excluded from the group. This task can be taken by the group controller. Note that even in the case of static policies, policy reconciliation cannot be avoided when several groups need to be merged.

We now discuss what happens when two or more groups need to be merged. If the groups to be merged have the origins in the same group – e. g. they are the result of a network partition that separated a group – and if the group policy is static, the groups should in fact have the same policy so no reconciliation will be necessary. What needs to be addressed is who will become the new group controller, since each policy specifies the same group creator of the original group, but different controllers.

Another case is when groups with the same name were created independently in partitioned components. Some systems uniquely identify groups based only on the group name, so they will try to merge the groups, which, can possibly have different policies. Again, there is no guarantee that a reconciliation is possible. In case a reconciliation is not possible, the servers can decide to destroy the group and inform all clients that the group was destroyed because a policy reconciliation was not possible. If the GCS identifies groups not only by name [2], then groups created independently in partitioned components will be interpreted as different groups and no merge and policy reconciliation will be required.

From the previous scenarios it is apparent that the policy framework should specify and provide support for the selection of a new group controller. There are several events that can drive such a need:

- a client or server crashed: The client that crashed was the group controller, or the server that crashed was serving the group controller [3].
- a network partition occurred: The group controller will end up only in one network component, while the other components will need to select a new group controller.
- a network merge occurred and policy reconciliation was possible: In this case the new merge

[2] One possibility is to add also the identifier of the server that represents the entire configuration of servers in a network component.

[3] Our failure model assumes that clients are not redirected when the server they are connected to crashes, so all the clients connected to that server will fail too.

group will have to select one of the groups to be merged controller, as the new group controller.

While we want the GCS to make the decisions, we would like to provide the application with the ability to specify the policy. Defining how failures should be handled can be done by the application. Of course some default policies can be used, in case an application does not want to deal with it. Faults can affect clients as well as servers, so a failure handling policy should be defined for both clients and servers.

Below we argue why a failure handling policy is required for both clients and servers. Consider the case of selecting a new group controller. If a group controller already exists, changing the group controller can be achieved by a simple role delegation. In case a group is merged, several legitimate group controllers will exist (one for each subgroup), the "oldest" controller will be selected as the new group controller.

An interesting case is when the group controller failed and there is no authority that can perform the role delegation. In this case we can define an extension of the role of the client as a group controller to the server that he is connected to, so the server can temporarily take over the role of the group controller and just deterministically select (acting as a delegator) a new group controller from a list provided by the application. If the application did not provide such a list, this will be perceived as a fatal failure and the server can just decide destroying the group.

Now, consider that the server itself crashed. In this case, the set of servers must decide which one of them will take over the task of selecting the new group controller. This can be done in several ways, the easiest is for example to deterministically select any of the servers (let's say the first). If the application wants to restrict this to a particular set of servers, it can provide an ordered set of potential take-over servers or a percentage if a voting policy is desired.

5 Related Work

There are several group communication systems that considered access control. The Ensemble secure group communication system [24, 25] assumes the 'fortress' model where an attack can come only from outside. The system uses a symmetric-key based key distribution scheme and uses Access Control List (ACL) as access control mechanism. The ACL is treated as replicated data within the group.

In [2] access control in groups is provided by using an authorization service, Akenti [27], which relies on X509 [22]. The method used is to have all group members registering with the authorization service offline to obtain a membership certificate signed by the Akenti server, and then when the group membership changes, every member verifies the membership certificate and the personal certificate of every member. The approach relies on identity for access control and provides a coarse granularity for access control.

Relevant to our work, but somehow orthogonal is the Antigone [17] framework. Antigone provides a policy framework that allows flexible application-level group security policies in a more relaxed model than the one usually provided by group communication systems. Also relevant to our work is [11] that defines general requirements and components for a secure group policy.

Most of the systems described above provide access control based on identity of participants and do not discuss how failures can affect the enforcement of policies. As oppose to above described schemes our approach is not identity-based. Instead, we take advantage of role-based access control [26, 10] and RT [15], a family of Role-based Trust-management languages, to define a fine-grained access control framework for group communication systems. Such systems have both scalability and fault-tolerance requirements. We reasoned about how these requirements can be met while providing flexibility to the application in defining specific policies.

6 Conclusions

In this paper we have analyzed the requirements access control mechanisms must fulfill in the context of group communication and defined a framework for supporting fine-grained access control for groups. Our framework combines role-based access control mechanisms with environment parameters (time, IP address, etc.) to provide policy support for a wide range of applications with very different requirements. In order to provide both flexible policy and efficient enforcement, we use the group communication servers to decide and enforce access control. We identify the set of all possible group operations that can be controlled and define the group policy as a mapping between roles and operations using context as constraints. In addition, we suggest a way in which failure policy can also be specified by the application.

Several things remain to be addressed in future work. They include: providing a "user-friendly" interface for our framework so that policies can be generated in an automatic way based on user specifications

and designing and implementing a parser engine that can translate application specific policies in system-understandable policies.

References

[1] *The TLS Protocol Version 1.0*. Number 2246 in RFC. T. Dierks and C. Allen, 1999. http://www.faqs.org/rfcs/rfc2246.html.

[2] D. A. Agarwal, O. Chevassut, M. R. Thompson, and G. Tsudik. An integrated solution for secure group communication in wide-area networks. In *Proceedings of the 6^{th} IEEE Symposium on Computers and Communications*, Hammamet, Tunisia, July 2001.

[3] Yair Amir, Claudiu Danilov, Michal Miskin-Amir, John Schultz, and Jonathan Stanton. The Spread toolkit: Architecture and performance. Technical report, 2004. CNDS-2004-1, Johns Hopkins University.

[4] Yair Amir, Danny Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*, pages 76–84, 1992.

[5] Yair Amir, Yongdae Kim, Cristina Nita-Rotaru, John Schultz, Jonathan Stanton, and Gene Tsudik. Secure group communication using robust contributory key agreement. In *To appear in Transactions on Parallel and Distributed Systems*, September 2003.

[6] Yair Amir, L. E. Moser, P. M. Melliar-Smith, D.A. Agarwal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems*, 13(4):311–342, November 1995.

[7] Yair Amir, Cristina Nita-Rotaru, Jonathan Stanton, and Gene Tsudik. Scaling secure group communication systems: Beyond peer-to-peer. In *the 3^{rd} DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, D.C., April 2003.

[8] Yair Amir and Jonathan Stanton. The Spread wide area group communication system. Technical Report 98-4, Johns Hopkins University, Center of Networking and Distributed Systems, 1998.

[9] Keneth P. Birman and Robert V. Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, March 1994.

[10] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control*. Artech House, April 2003.

[11] H. Harney, A. Colegrove, and P. McDaniel. Principles of policy in secure groups. In *Network and Distributed Systems Security*, San Diego, CA, February 2001.

[12] Hugh Harney, Andrea Colegrove, and Patrick McDaniel. Principles of policy in secure groups. In *Network and Distributed Systems Security Symposium*, 2001.

[13] Kim Potter Kihlstrom, Louise E. Moser, and P. M. Melliar-Smith. The SecureRing protocols for securing group communication. In *Proceedings of the IEEE 31st Hawaii International Conference on System Sciences*, pages 317–326, January 1998.

[14] John Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (Version 5). RFC-1510, September 1993.

[15] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[16] P. McDaniel and A. Prakash. Methods and limitations of security policy reconciliation. In *IEEE Symposium on Security and Privacy*, pages 73–87, Oakland, CA, May 2002.

[17] Patrick McDaniel, Atul Prakash, and Peter Honeyman. Antigone: A flexible framework for secure group communication. In *Proceedings of the 8th USENIX Security Symposium*, pages 99–114, August 1999.

[18] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, pages 33–38, September 1994.

[19] Cristina Nita-Rotaru and Ninghui Li. A framework for role-based access control in group communication systems. Technical report, 2003. CERIAS TR-2003-31, Purdue University.

[20] Michael K. Reiter. Secure agreement protocols: reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80. ACM, November 1994.

[21] R. V. Renesse, K.Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39:76–83, April 1996.

[22] ITU-T Rec. X.509 (revised). *The Directory - Authentication Framework*. International Telecommunication Union, 1993.

[23] O. Rodeh, K. Birman, and D. Dolev. Optimized group rekey for group communication systems. In *Proceedings of ISOC Network and Distributed Systems Security Symposium*, February 2000.

[24] Ohad Rodeh, Ken Birman, and Danny Dolev. Using AVL trees for fault tolerant group key management. Technical Report 2000-1823, Cornell University, Computer Science; Tech. Rep. 2000-45, Hebrew University, Computer Science, 2000.

[25] Ohad Rodeh, Ken Birman, and Danny Dolev. The architecture and performance of security protocols in the Ensemble Group Communication System. *ACM Transactions on Information and System Security*, 4(3):289–319, August 2001.

[26] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[27] Mary R. Thompson, Abdelilah Essiari, and Srilekha Mudumbai. Certificate-based authorization policy in a PKI environment. *ACM Trans. Inf. Syst. Secur.*, 6(4):566–588, 2003.

[28] B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered multicast protocol. In *Theory and Practice in Distributed Systems, International Workshop*, LNCS, page 938, September 1994.

[29] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. In *Proceedings of the ACM SIGCOMM '98*, pages 68–79, 1998.