Framework for Authentication and Access Control of Client-Server Group Communication Systems *

Yair Amir, Cristina Nita-Rotaru, and Jonathan R. Stanton

Department of Computer Science
Johns Hopkins University
3400 North Charles St.
Baltimore, MD 21218 USA
{yairamir, crisn, jonathan}@cs.jhu.edu

Abstract. Researchers have made much progress in designing secure and scalable protocols to provide specific security services, such as data secrecy, data integrity, entity authentication and access control, to multicast and group applications. However, less emphasis has been put on how to integrate security protocols with modern, highly efficient group communication systems and what issues arise in such secure group communication systems. In this paper, we present a flexible and modular architecture for integrating many different authentication and access control policies and protocols with an existing group communication system, while allowing applications to provide their own protocols and control the policies. This architecture maintains, as much as possible, the scalability and performance characteristics of the unsecure system. We discuss some of the challenges when designing such a framework and show its implementation in the Spread wide-area group communication toolkit.

1 Introduction

The Internet is used today not only as a global information resource, but also to support collaborative applications such as voice- and video-conferencing, whiteboards, distributed simulations, games and replicated servers of all types. Such collaborative applications often require secure message dissemination to a group and efficient synchronization mechanisms. Secure group communication systems provide these services and simplify application development.

A secure group communication system needs to provide confidentiality and integrity of client data, integrity, and possibly confidentiality, of server control data, client authentication, message source authentication and access control of system resources and services.

Many protocols, policy languages and algorithms have been developed to provide security services to groups. However, there has not been enough study

* This work was supported by grant F30602-00-2-0526 from The Defense Advanced Research Projects Agency.

of the integration of these techniques into group communication systems. Needed is a scheme flexible enough to accommodate a range of options and yet simple and efficient enough to appeal to application developers. Complete secure group communication systems are very rare and research on how to transition protocols into complete systems has been scarce.

Secure group systems really involve the intersection of three major, and distinct, research areas: networking protocols, distributed algorithms and systems, and cryptographic security protocols.

A simplistic approach when building a secure group system is to select a specific key management protocol, a standard encryption algorithm, and an existing access control policy language and integrate them with a messaging system. This would produce a working system, but would be complex, fixed in abilities, and hard to maintain as security features would be mixed with networking protocols and distributed algorithms.

In contrast, a more sophisticated approach is to construct an architecture that allows applications to plug-in both their desired security policy *and* the mechanisms to enforce the policy. Since each application has its particular security policies, it is natural to give an application more control not only on specifying the policy, but on the implementation of the services part of the policy too.

This paper proposes a new approach to group communication system architecture. More precisely, it provides such an architecture for authentication and access control. The architecture is flexible, allowing many different protocols to be supported and even be executing at the same time; it is modular so that security protocols can be implemented and maintained independently of the network and distributed protocols that make up the group messaging system; it allows applications to control what security services and protocols they use and configure; it efficiently enforces the chosen security policy without unduely impacting the messaging performance of the system.

As many group communication systems are built around a client-server architecture where a relatively small number of servers provide group communication services to numerous clients, we focused on systems utilizing this architecture. [1]

We implemented the framework in the Spread wide-area group communication system. We evaluate the flexibility and simplicity of the framework through six case studies of different authentication and access control methods. We show how both simple (IP based access control, password based authentication) and sophisticated (SecurID, PAM, anonymous payment, and group based) protocols can be supported by our framework.

Note that this paper is *not* a defense of any particular access control policy, authentication method or group trust model. Instead, it provides a flexible, complete interface to allow many such polices, methods, or models to be expressed and enforced by an existing, actively used group communication system.

The rest of the paper is organized as follows. Section 2 overviews related work. We present the authentication and access control framework and its im-

[1] Some of the work may apply to network level multicast, but we have not explored that.

plementation in the Spread toolkit in Section 3. We provide several brief case studies of how diverse protocols and policies can be supported by the framework in Section 4. Finally, we conclude and discuss future directions.

2 Related Work

There are two major directions in secure group communication research. The first one aims to provide security services for IP-Multicast and reliable IP-Multicast. Research in this area assumes a model consisting of one sender and many receivers and focuses on the high scalability of the protocols. Since the presence of a shared secret can be used as a foundation of efficiently providing data confidentiality and data integrity, a lot of work has been done in designing very scalable key management protocols. For lack of space we cite only the very recent ones: the VersaKey Framework [10] and the Group Secure Association Key Management Protocol (GSAKMP) [12].

The second major direction in secure group communication research is securing application level multicast systems, also known as group communication systems. These systems assume a many-to-many communication model where each member of the group can be both a receiver and a sender, and provide reliability, strong message ordering and group membership guarantees, with moderate scalability. Initially group communication systems were designed as high-availability, fault-tolerant systems, for use in local area networks. Therefore, the first group communication systems ISIS [9], Horus [21], Transis [4], Totem [5], and RMP [25] were less concerned with addressing security issues, and focused more on the ordering and synchronization semantics provided to the application (the Virtual Synchrony [8] and Extended Virtual Synchrony [17] models).

The number of secure group communication systems is small. Besides our system (Spread), the only implementation of group communication systems that focus on security are the RAMPART system at AT&T [20], the SecureRing [14] project at UCSB and the Horus/Ensemble work at Cornell [22]. A special case is the Antigone [16] framework, designed to provide mechanisms allowing flexible application security policies. Most relevant to this work are the Ensemble and the Antigone systems. Ensemble focused on optimizing group key distribution, and chose to allow application-dependent trust models in the form of access control lists treated as replicated data within the group. Authentication is achieved by using PGP. Antigone instead, allows flexible application security policies (rekeying policy, membership awareness policy, process failure policy and access control policy). However, it uses a fixed protocol to authenticate a new member and negotiate a key, while access control is performed based on a pre-configured access control list.

We also consider frameworks designed with the purpose of providing authentication and/or access control, without addressing group communication issues. Therefore, they are complementary to our work. One of these frameworks is the Pluggable Authentication Module (PAM) [23] which provides authentication services to UNIX system services (like login, ftp, etc). PAM allows an application

not only to choose how to authenticate users, but also to switch dynamically between the authentication mechanisms without (rewriting and) recompiling a PAM-aware application. Other frameworks providing access control and authentication services are systems such as Kerberos [15] and Akenti [24]. Both of them have in common the idea of authenticating users and allowing access to resources, with the difference being that Kerberos uses symmetric cryptography, while Akenti uses public-key cryptography to achieve their goals.

One flexible module system that supports various security protocols is Flexinet [13]. Flexinet is an object oriented framework that focuses on dynamic negotiations, but does not provide any group-oriented semantics or services.

3 General System Architecture

The overall goal of this work is to provide a framework that integrates many different security protocols and supports all types of applications which have changing authentication and access control policy requirements, while maintaining a clear separation of the security policy from the group messaging system implementation. In this section, after discussing some design considerations, we present the authentication and access control frameworks.

3.1 Why is a General Framework Needed?

When a communication system may only be used with one particular application, integrating the specific security policy and needed protocols with the system may make sense. However, when a communication system needs to support many different applications that may not always be cooperative, separating the policy issues which will be unique to each application from the enforcement mechanisms which must work for all applications avoids an unworkable "one-size-fits-all" security model, while maintaining efficiency.

Separating the policy implementation from both the application and the group communication system is also useful because in a live, production environment, the policy restrictions and access rules will change much more often than the code or system changes. So modifications of policy modules should not require recompiling or changing the application code.

The features of the general framework, as opposed to the features of a particular authentication or access control protocol, are:

1. Individual policies for each application.
2. Efficient policy enforcement in the messaging system.
3. Simple interface for both authentication and access control modules.
4. Independence of the messaging system from security protocols.
5. Many policies and protocols work with the framework, including: access control lists, password authentication, public/private key, certificates, role based access control, anonymous users, and dynamic peer-group policies.

We distinguish between authentication and access control modules to provide more flexibility. Each type of module has a distinctive interface which supports its specific task. The authentication module verifies that a client is who it claims to be. The access control module decides about all of the group communication specific actions a client attempts after it has been authenticated: join or leave a group, send an unicast message to another client or multicast a message to a group. It also decides whether a client is allowed to connect to a server (the access control module can deny a connection even if the authentication succeeded).

The framework supports *dynamic* policies. The main challenge with such policies is to allow changes during execution. Since the framework itself does not have any knowledge of the actual policy, for example it does not cache decisions or restrict what form actual policies take, it is possible for the access control modules to change how they make decisions independently of server. The modules need to make sure they activate dynamic changes in a consistent way, by using synchronized clocks, or by using the group communication services to agree on when to activate changes.

3.2 Framework Implementation in Spread

We implemented the framework in the Spread group communication system to give a concrete, real-world basis for evaluating the usefulness of this general architecture. Although we only implemented the framework within the Spread system, the model and the interface of the framework are actually quite general and the set of events upon which access control decisions can be made includes all of the available actions in a group-based messaging service (join, leave, group send, unicast send, connect).

3.3 The Spread Group Communication Toolkit

Spread [7,3,2] is a local and wide-area messaging infrastructure supporting reliable multicast and group communication. It provides reliability and ordering of messages (FIFO, causal, total ordering) and a membership service. The toolkit supports four different semantics: No membership, Closely Synchronous[2], Extended Virtual Synchrony (EVS) [1] and View Synchrony (VS) [11].

The system consists of one or more servers and a library linked with the application. The servers maintain most of the state of the system and provide reliable multicast dissemination, ordering of messages and the membership services. The library provides an API and basic services for message oriented applications. The application and the library can run on the same machine as a Spread server, in which case they communicate over IPC, or on separate machines, in which case the client-server protocol runs over TCP/IP.

Note that in order to implement our framework, we needed to modify both the Spread client library and the Spread daemon. When an application implements its own authentication and access control method, it needs to implement both

[2] This is a relaxed version of EVS for reliable and FIFO messages.

the client side and the server side modules, however, it does not need to modify the Spread library or the Spread daemon.

In Spread each member of the group can be both a sender and a receiver. The system is designed to support small to medium size groups, but can accommodate a large number of different collaboration sessions, each of which spans the Internet. This is achieved by using unicast messages over the wide-area network and routing them between Spread nodes on an overlay network. Spread scales well with the number of groups used by the application without imposing any overhead on the network routers. Group naming and addressing is not a shared resource (as in IP multicast addressing), but rather a large space of strings which is unique to a collaboration session.

The Spread toolkit is available publicly and is being used by several organizations for both research and practical projects. The toolkit supports cross-platform applications and has been ported to several Unix platforms as well as Windows and Java environments.

3.4 Authentication Framework

All clients are authenticated when connecting to a server, and trusted afterwards. Therefore, when a client attempts actions, such as sending messages or joining groups, no authentication is needed. However, the attempted user actions are checked against a specified policy which controls which actions are permitted or denied for that user. This approach explicitly assumes that as long as a connection to the server is maintained, the same user is authenticated.

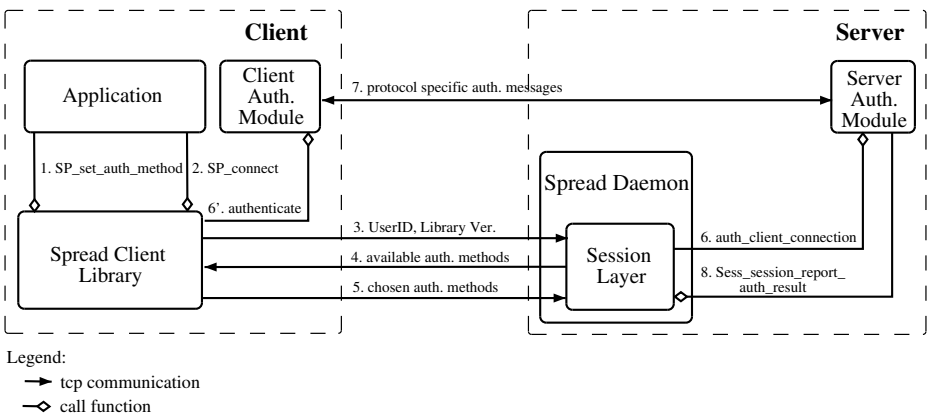

Fig. 1. Authentication architecture and communication flow

Figure 1 presents the architecture and the process of authentication. Both the client and the server implement an authentication module.

The change on the client side consists of the addition of a function (see Figure 2) that allows an application to set the authentication protocol it wishes to use and to pass in any necessary data to that protocol, before connecting

to a Spread server. When the function that specifies the request of a client to connect to a server is called (`SP_connect`), the connection tries to use the method the application set to establish a connection. The authentication method chosen by the application applies to all connections established by this application.

```
1  int SP_set_auth_method( const char *auth_name, int (*authenticate) (int, void *), void * auth_data );
2  int SP_set_auth_methods( int num_methods, const char *auth_name[], int (*authenticate[]) (int, void *), void * auth_data[] );
3
4  /* declaration of authenticate function */
5  int authenticate(int fd, void * user_data_pointer);
```

Fig. 2. Client Authentication Module API

A server authentication module needs to implement the functions listed in the `auth_ops` structure (see Figure 3, line 10). Then the module should register itself with the Spread daemon by calling the `Acm_auth_add_method` function. By default, a module is registered in the 'disabled' state. The system administrator can enable the module when configuring Spread.

```
1  struct session_auth_info {
2      int ses;
3      void *module_data;
4      int num_required_auths;
5      int completed_required_auths;
6      int required_auth_methods[MAX_AUTH_METHODS];
7      int required_auth_results[MAX_AUTH_METHODS];
8  };
9
10 struct auth_ops {
11     void (*auth_client_connection) (struct session_auth_info *sess_auth_p);
12 };
13
14 struct acp_ops {
15     bool (*open_connection) (char *user);
16     bool (*open_monitor) (char *user); /* not used currently */
17     bool (*join_group) (char *user, char *group, void *acm_token);
18     bool (*leave_group) (char *user, char *group, void *acm_token);
19     bool (*p2p_send) (char *user, char dests[][MAX_GROUP_NAME], int service_type);
20     bool (*mcast_send) (char *user, char groups[][MAX_GROUP_NAME], int service_type);
21 };
22
23 /* Auth Functions */
24 bool Acm_auth_add_method(char *name, struct auth_ops *ops);
25
26 /* Access Control Policy Functions */
27 bool Acm_acp_set_policy(char *policy_name);
28 bool Acm_acp_add_method(char *name, struct acp_ops *ops);
```

Fig. 3. Server Authentication and Access Control Module API

The authentication process begins when the session layer of the daemon receives a connection request from a client. After some initial information exchange and negotiation of the allowed authentication protocols, the session module constructs a `session_auth_info` structure containing the list of agreed upon authentication protocols. This structure is passed as a parameter to each authentication function and is used as a handle for the entire process of authenticating a client. The authentication function can use the `module_data` pointer to store any module specific data that it needs during authentication. The session layer calls the `auth_client_connection` method for each protocol and then "forgets about" the client connection. A minimal state about the client is stored, but no messages are received or delivered to the client at this point.

The `auth_client_connection` function is responsible for authenticating the client connection. If authenticating the client will take a substantial amount of

CPU or real time, the function should not do the work directly, but rather setup a callback function to be called later (for example when messages arrive from the client), and then it should return. Another approach is to fork off another process to handle the authentication. This is required because the daemon is blocked while this function is running.

The `auth_client_connection` function never returns a decision value because a decision may not have been reached yet. When a decision has been made the server authentication module calls `Sess_session_report_auth_result` and releases control to the session layer. The `Sess_session_report_auth_result` function reports whether the current authentication module has successfully authenticated the session or not. If more than one authentication method was required, the connection succeeds if all the methods succeed.

3.5 Access Control Framework

In our model, an authenticated client connection is not automatically allowed to perform any actions. Each action a client may request of the server, such as sending a message or joining or leaving a group, is checked at the time it is attempted against an access control policy module. The enforcement checks are implemented by having the session layer of the server call the appropriate access control policy module callback function (see Figure 3, lines 14-20) return a decision. The implementation of the check functions should be optimized as they have a direct impact on the performance of the system as they are called for every client action.

If the module chooses to allow the request, then the server handles it normally. In the case of rejection, the server creates a special "reject" message which will be sent to the client in the normal stream of messages. The reject message contains as much of the data included in the original attempt as possible. The application should be able to identify which message was rejected by whatever information it stored in the body of the message (such as an application level sequence number) and respond to it appropriately. That response could be a notification to the user, establishing a new connection with different authentication credentials and retrying the request, logging an error, etc.

The server can reject an action at two points, when the server receives the action from the client or when the action is going to take effect. For example, when a client joins a group the join can be rejected when the join request is received from the directly connected client, and when the join request has been sent to all of the servers and has been totally ordered. Rejecting the request the first time it is seen avoids processing requests that will later be rejected and simplifies the decision-making because only the server the client is directly connected to will make the decision. The disadvantage is that at the time the request is being accepted or rejected the module only knows the current state of the group or system and not what the state will be when the request would be acted upon by the servers. Since these states can differ, some type of decisions may not be possible at the early decision point.

4 Case Studies

To provide some intuition as to what building a Spread authentication module requires, this section discusses the implementation of several real-world modules: an IP based access control module, a password based authentication module, a SecurID or PAM authentication module, an anonymous payment authentication and anonymous access control module, and a dynamic peer-group authentication module. For more details and implementation code see [6].

IP Access Control. A very simple access control method that does not involve any interaction with the client process or library, is one that is based on the IP address of the clients. The connection is allowed considering the IP address from which the client connected to the server. This module only restricts the `open_connection` (see Figure 3, line 15) operation.

Password Authentication. A common form of authentication uses some type of password and username to establish the identity of the user. Many types of password based authentication can be supported by our framework from passwords sent in the clear (like in telnet) to challenge-response passwords.

To implement a password-based authentication method, both a client and a server side need to be implemented. The server module can use the Events subsystem in Spread to wait for network events to occur and avoid blocking the server while the user is entering its password or the client and server modules are communicating. The client module consists of one function which is called during the establishment of a connection and returns either success or failure. The function can use the file descriptor of the socket over which the connection is being established and whatever data pointer was registered by the `SP_set_auth_method`. In this case the application prompted the user for a username and password and created a `user_password` structure. The `authenticate` function, sends the username and the password to the server and waits for a response, informing it of whether or not the authentication succeeded.

SecurID. A popular authentication method is RSA SecurID. The method uses a SecurID server to authenticate a SecurID client based on a unique randomly generated identifier and a PIN. In some cases the SecurID server might ask the client to provide new credentials. We do not discuss here the internal of the SecurID authentication mechanism (see [19] for more details), but focus on how our framework can accommodate this method.

The main difference from the previous examples is that here the Server Authentication Module needs to communicate with the SecurID server. As mentioned before, the `auth_client_connection` function should not block. Blocking can happen when opening a connection with a SecurID server and retrieving messages from it. Therefore, `auth_client_connection` forks another process responsible for the authentication protocol and then registers an event such that it will get notified when the forked process finished. The forked process establishes a connection with the SecurID Server and authenticates the user. When it finishes, the Server Authentication Module gets notified, so it can call the `Sess_session_report_auth_result` function to inform the Spread daemon that a decision was taken and to pass control back to it.

PAM. Another popular method of authentication is the modular PAM [23] system which is standard on Solaris and many Linux systems. Here the authentication module will act as a client to a PAM system and request authentication through the standard PAM function calls. To make authentication through PAM work, the module must provide a way for PAM to communicate and interact with the actual human user of the system, to prompt for a password or other information. The module would register an interactivity function with PAM that would pass all of the requests to write to the user or request input from the user over the Spread communication socket to the Spread client authentication module for PAM. This client module would then act on the PAM requests and interact with the user and then send the reply back to the Spread authentication module which would return the results to the actual PAM function.

Anonymous payments. An interesting approach is when access is provided to anonymous clients in exchange for payment. These systems [18] perform transactions between a client and a merchant, assuming that both of them have accounts with a Bank. By using cryptographic techniques, the system provides anonymity of the client and basic security services. We do not detail the cryptographic details, but show how this method can be accommodated in our framework.

We assume support from the anonymous payments system (in the form of an API) and require the servers and the client to have an account with a Bank. When a client connects to a server, the Client Authentication Module generates a check and an identifier of client's account and then passes them to the Server Authentication Module which will then contact the Bank to validate the check (if necessary another process will be forked as in the SecurID case). When validated, the Server Authenticated Module will register the client's identifier with the access control policy as a paid user of the appropriate groups. Then, for as long as the payment was valid, the client will be permitted to access the groups they paid for and the server has no knowledge of the client's identity.

Group-Based Authentication. In all the previous authentication methods presented, the authentication of a client is handled by the server that the client connects to. In larger, non-homogeneous environments authentication may involve some or all of the group communication system servers. Although these protocols may be more complex, they can provide better mappings of administrative domains, and possibly better scalability.

An example of such a protocol is when a server does not have sufficient knowledge to check a client's credentials (for instance a certificate). In this case, it sends the credentials to all the servers in the configuration and each server then attempts to check the credentials itself and sends an answer back. If at least one server succeeds, the client is authenticated. The particularity of such a protocol is that the servers need to communicate between them as part of the authentication process. Since all the servers can communicate between them in our system, the framework provides all necessary features that allows the integration of such a group-based authentication method.

Access Control. We realize that the above case studies are focused on authentication. Few standard access control protocols that we could use as case studies

exist. To demonstrate the ability of the access control architecture we create a case study about an imaginary secure IRC system. Consider a set of users where some users are allowed to chat on the intelligence group, while others are restricted to the operations group. Some are allowed to multicast to a group but are not allowed to read the group messages (virtual drop-box). Our framework supports these access control policies through appropriate implementation of the join and multicast hooks defined in Figure 3. Access control modules support identity based, role based, or credential based restrictions.

5 Conclusions and Future Work

We presented a flexible implementation of an authentication and access control framework in the Spread wide area group communication system. Our approach allows an application to write its own authentication and access control modules, without needing to modify the Spread client or server code. The flexibility of the system was showed by showing how a wide range of authentication methods can be implemented in our framework.

There are a lot of open problems that are subject of future work. These include: providing tools that allow an application to actually specify a policy, handling policies in a system supporting network partitions (for example merging components with different policies), providing support for meta-policies defining which entity is allowed to create or modify group policies, and developing dynamic group trust protocols for authentication.

References

1. Amir, Y. *Replication using Group Communication over a Partitioned Network*. PhD thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1995.
2. Amir, Y., Awerbuch, B., Danilov, C., and Stanton, J. Flow control for many-to-many multicast: A cost-benefit approach. Tech. Rep. CNDS-2001-1, Johns Hopkins University, Center of Networking and Distributed Systems, 2001.
3. Amir, Y., Danilov, C., and Stanton, J. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proceedings of the International Conference on Dependable Systems and Networks* (June 2000), pp. 327–336.
4. Amir, Y., Dolev, D., Kramer, S., and Malki, D. Transis: A communication sub-system for high availability. *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems* (1992), 76–84.
5. Amir, Y., Moser, L. E., Melliar-Smith, P. M., Agarwal, D., and Ciarfella, P. The totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems 13*, 4 (November 1995), 311–342.
6. Amir, Y., Nita-Rotaru, C., and Stanton, J. Framework for authentication and access control of client-server group communication systems. Tech. Rep. CNDS 2001-2, Johns Hopkins University, Center of Networking and Distributed Systems, 2001. http://www.cnds.jhu.edu/publications/.
7. Amir, Y., and Stanton, J. The Spread wide area group communication system. Tech. Rep. 98-4, Johns Hopkins University, Center of Networking and Distributed Systems, 1998.

8. Birman, K. P., and Joseph, T. Exploiting virtual synchrony in distributed systems. In *11th Annual Symposium on Operating Systems Principles* (November 1987), pp. 123–138.
9. Birman, K. P., and Renesse, R. V. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, March 1994.
10. Caronni, G., Waldvogel, M., Sun, D., Weiler, N., and Plattner, B. The VersaKey framework: Versatile group key management. *IEEE Journal of Selected Areas in Communication 17*, 9 (September 1999).
11. Fekete, A., Lynch, N., and Shvartsman, A. Specifying and using a partitionable group communication service. In *Proceedings of the 16th annual ACM Symposium on Principles of Distributed Computing* (Santa Barbara, CA, August 1997), pp. 53–62.
12. Harney, H., Colegrove, A., Harder, E., Meth, U., and Fleischer, R. Group secure association key management protocol (GSAKMP). draft-irtf-smug-gsakmp-00.txt, November 2000.
13. Hayton, R., Herbert, A., and Donaldson, D. FlexiNet — A flexible component oriented middleware system. In *Proceedings of SIGOPS'98* (url-http://www.ansa.co.uk/, 1998).
14. Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. The SecureRing protocols for securing group communication. In *Proceedings of the IEEE 31st Hawaii International Conference on System Sciences* (Kona, Hawaii, January 1998), vol. 3, pp. 317–326.
15. Kohl, J., and Neuman, B. C. The Kerberos Network Authentication Service (Version 5). RFC-1510, September 1993.
16. McDaniel, P., Prakash, A., and Honeyman, P. Antigone: A flexible framework for secure group communication. In *Proceedings of the 8th USENIX Security Symposium* (August 1999), pp. 99–114.
17. Moser, L. E., Amir, Y., Melliar-Smith, P. M., and Agarwal, D. A. Extended virtual synchrony. In *Proceedings of the IEEE 14th International Conference on Distributed Computing Systems* (June 1994), IEEE Computer Society Press, Los Alamitos, CA, pp. 56–65.
18. Neuman, B. C., and Medvinsky, G. Requirements for network payment: The netcheque perspective. In *In Proceedings of IEEE COMPCON'95* (March 1995).
19. Nystrom, M. The SecurID SASL mechanism. RFC-2808, April 2000.
20. Reiter, M. K. Secure agreement protocols: reliable and atomic group multicast in RAMPART. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security* (November 1994), ACM, pp. 68–80.
21. Renesse, R. V., K.Birman, and Maffeis, S. Horus: A flexible group communication system. *Communications of the ACM 39* (April 1996), 76–83.
22. Rodeh, O., Birman, K., and Dolev, D. The architecture and performance of security protocols in the Ensemble Group Communication System. *ACM Transactions on Information and System Security* (To appear).
23. Samar, V., and Schemers, R. Unified login with Pluggable Authentication Modules (PAM). OSF-RFC 86.0, October 1995.
24. Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K., and Essiari, A. Certificate-based access control for widely distributed resources. In *Proceedings of the Eighth Usenix Security Symposium* (August 1999), pp. 215–227.
25. Whetten, B., Montgomery, T., and Kaplan, S. A high performance totally ordered multicast protocol. In *Theory and Practice in Distributed Systems, International Workshop* (September 1994), Lecture Notes in Computer Science, p. 938.