CITI Technical Report 99-2

Antigone: A Flexible Framework for Secure Group Communication

Patrick McDaniel
pdmcdan@eecs.umich.edu
Atul Prakash
aprakash@eecs.umich.edu
Peter Honeyman
honey@citi.umich.edu

ABSTRACT

Many emerging applications on the Internet requiring group communication have varying security requirements. Significant strides have been made in achieving strong semantics and security guarantees within group environments. However, in existing solutions, the scope of available security policies is often limited. This paper presents Antigone, a framework that provides a suite of mechanisms from which flexible application security policies may be implemented. With Antigone, developers may choose a policy that best addresses their security and performance requirements. We describe the Antigone's mechanisms, consisting of a set of micro-protocols, and show how different security policies can be implemented using those mechanisms. We also present a performance study illustrating the security/performance tradeoffs that can be made using Antigone.

September 8, 1999

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

Antigone: A Flexible Framework for Secure Group Communication

Patrick McDaniel* Atul Prakash
EECS Dept.
University of Michigan
Ann Arbor
aprakash@eecs.umich.edu

Peter Honeyman
Center for Information Technology Integration
University of Michigan
Ann Arbor
honey@citi.umich.edu

1 Introduction

IP multicast [Dee89] services are becoming more widely available on the Internet. The use of group communication based on IP multicast as a fundamental building block for a variety of applications such as video conferencing will increase. Many interesting applications that require group communication, such as video conferencing, collaborative applications, data casting, and virtual communities, are emerging on the Internet. These applications, depending on the perceived risks and performance requirements, require different levels of security. In this paper, we present a system, called Antigone, that provides mechanisms for building a range of security policies for group communication.

Existing approaches provide two predominant policies. Secure multicast systems [HM97, KA98] view groups as collections of participants that require a shared secret, called a *session key*, to secure application traffic. Group membership may change without affecting the security context. A potential security risk is that past members of the group may have access to current content. The advantage of this approach is low cost; rekeying after membership changes is not needed.

Conversely, secure group communication systems [Rei94] typically have threat models that require the changing of the security context after each membership change. Thus, protection from members not currently in the group is achieved at the cost of the context change. Some systems support stronger threat models than are strictly required by applications at a high performance cost.

The Antigone framework provides a flexible interface for the definition and implementation of a wide range of secure group policies. A central element of the Antigone architecture is a set of *mechanisms* that provide the basic services needed for secure groups. Policies are implemented by the composition and configuration of these mechanisms. Thus, Antigone does not dictate the available security policies to an application, but provides high-level mechanisms for implementing them.

To ease the task of application development, we also provide a facility for configuring rapidly a security policy from a set of standard policies that have been implemented using the Antigone mechanisms. An application is free to use these standard policies, if they are satisfactory for its purpose, or build its own using the high-level mechanisms provided by Antigone.

Antigone is targeted for systems likely to require IP-multicast services. The majority of existing multicast based systems distribute high bandwidth content to a highly dynamic membership. A natural requirement of these systems is for efficient group and security management.

We see videoconferencing as representative of the types of applications that may benefit from the Antigone framework. However, we do not limit Antigone to continuous media systems. A fully functional secure group communication

*This research is supported in part by the NASA Graduate Student Researchers Program, Kennedy Space Center, Grant Number 10-52613

system may be built on the Antigone mechanisms.

While there are well known techniques for providing secure groups, Antigone has several goals that make it unique. We state the following as the five primary goals of Antigone.

1. **Flexible Security Policy** - An application should be able to use a wide range of security policies, with appropriate performance tradeoffs.
2. **Flexible Threat Model** - The system should support threat models appropriate for a wide range of applications.
3. **Security Infrastructure Independence** - Our solution should not be dependent on the availability of a specific security infrastructure.
4. **Transport Layer Independence** - The solution should not depend on the availability of any single transport mechanism (such as IP Multicast [Dee89]). On the other hand, our solution should be able to take advantage of IP-multicasts when they are available.
5. **Performance** - The performance overheads of implementing security policies should be kept low.

An early version of Antigone has been integrated into the `vic` [Net96] videoconferencing system. The result of that effort, called the Secure Distributed Virtual Conferencing (SDVC) application, was used to broadcast securely the September 1998 Internet 2 Member Meeting to several sites across the United States. The broadcast and other WAN tests indicate the viability of our approach. Using high speed cryptographic algorithms we are able to attain television-like secure video frame rates (30 frames/second) over a LAN. Details of the implementation and our experiences deploying SDVC can be found in [MHP98, AAC+99].

The remainder of the paper is organized as follows. Section 2 maps the design space of secure group communication policies. Section 3 outlines existing designs and techniques of secure group communication. Section 4 outlines Antigone's high-level, layered architecture. Section 5 presents the mechanisms that can be used to implement a wide range of group security policies. Section 6 shows that a wide range of security policies can be implemented using these mechanisms. Section 7 presents the available multicast transport modes of Antigone. Section 8 presents preliminary performance results for some of the policies that can be implemented using Antigone. Section 9 summarizes our conclusions and presents directions for future work.

2 Group Security Policies

Different group communication applications require different security policies, depending on their threat model and performance requirements. In this section, we point out some of the important dimensions along which group security policies vary. To address the requirements of a wide range of applications, Antigone attempts to provide a basic set of high-level mechanisms that can be used to implement a range of group security policies. The dimensions that are discussed below are: *session rekeying policy*, *application message policy*, *membership awareness policy*, and *failure policy*. The session rekeying policy defines the reaction of the group to changes in membership in terms of rekeying sessions. The application message policy defines the security guarantees with which application messages are transmitted. The membership policy dictates the availability of membership information to members. The failure policy defines the type of failures handled by the system. The remainder of this section describes the nature and implications of these policy decisions.

Antigone assumes that all group members are trusted, i.e., members do not actively attempt to circumvent the security of the system. Non-members however may attempt to intercept messages, modify messages, or prevent messages from being delivered.

2.1 Session Rekeying Policy

A common strategy to support secure group communication among trusted members is to use a common symmetric session key. An important policy issue for a group communication application is deciding when a session must be rekeyed, i.e., the old session key is discarded and a new session key is sent to all the members.

There is a close relationship between session rekeying and group membership. Applications often need protection from members not in the current *view*.[1] Therefore, changes in group membership require the session to be rekeyed. If rekeying is not performed after each change in membership, the view will not reflect a secure group, but indicates only which members are actively participating in the session. Past members may retain the session key and continue to receive content. Future members may record and later decode current and past content. In applications that do not need protection from past or future members, rekeying after membership events is unnecessary.

We define the *view group* to be the set of members in the current group membership view. A *key group* is the set of members who have access to the current session key. The key group may be a subset, a superset, or overlap the view group.

We say that a group security policy is said to be *sensitive* to an event if it changes the security context in response to the reception of the event. Typically, the security context is changed by distributing a new session key (rekeying).

Group security policy is often sensitive to *group membership events*. Group membership events include (1) `JOIN` event, which is triggered when a member is allowed to join the group; (2) `LEAVE` events, which is triggered when a member leaves the group; (3) `PROCESS FAILURE` events, when a member is assumed to have failed in some manner; and (4) `MEMBER EJECT` events, when a previously admitted member is purged from the group according to some group policy. In addition, sessions may be rekeyed when a specified time interval has passed since the last session rekey.

The sensitivity of a policy directly defines its threat model. For example, consider a group model that is sensitive to only `MEMBER EJECT` events. Because the session is rekeyed after member ejection, the key group will never contain an ejected member. Thus, the application is assured that no ejected member will ever have access to future content. However, the session content is not protected from processes that have left voluntarily, are assumed to have failed, or join in the future.

The sensitivity mechanisms in Antigone can be used to build a large number of session rekeying policies. In the following text, we define and illustrate four general purpose policies that are representative of the policies implemented in existing systems.

Time-sensitive Rekeying Policy
Groups implementing a time-sensitive policy periodically rekey, independent of group membership events. Periodic rekeying prevents the session key from "wearing out". The group attempts to guard against cryptanalysis by using a session key only for a limited period.

By periodically rekeying, the group may be protected from an adversary who wishes to block the delivery of new session keys. An adversary blocking rekeying messages may intend for the group to continue to use the current session key. With a time-sensitive rekeying policy, if a new key is not successfully established after the current session key expires, group members can choose to no longer communicate rather than use the expired session key.

Time-sensitive rekeying can be useful in a secure on-line subscription service. Paying members would periodically be sent a new key that is valid until the next subscription interval. The GKMP [HM97] protocol implements a time-sensitive rekeying policy.

Typically, systems implementing time-sensitive groups use *Key Encrypting Keys* (KEK) [HM97] to reduce the costs of rekeying. In systems that use KEKs, the key group contains all previous and current members of the group. This approach is limited in that any member may continue to access the group content after leaving. Systems that use KEKs cannot forcibly eject members without additional infrastructure.

All rekeying in Antigone is *session key independent*. Session key independence states that knowledge of one session key provides no information about others. With this independence, Antigone provides a stronger guarantee than time sensitive groups that use KEKs; a member who has left the group may continue to access the group content only until the next rekey. A past member cannot access current or future group content without again joining as a member.

Leave-Sensitive Rekeying Policy
Groups implementing a leave-sensitive policy rekey after `LEAVE`, `PROCESS FAILURE`, and `MEMBER EJECT` events.

[1] A group *view* is the set of identities associated with the members of the group during a period where no changes in membership occur. If the membership changes (a member joins or leaves the group), then a new view is created. This is a similar concept to Birmans's group view [Bir93].

Thus, the key group may be arbitrarily larger than the view group.

The threat model implied by leave sensitive groups states that any member who has left the group will not have access to current or future content. For example, a business conferencing system that supports negotiations between a company's representatives and a supplier may benefit from leave-sensitive rekeying. Once the supplier leaves, a leave-sensitive rekey policy would prevent subsequent discussions from being available to the supplier, even if the supplier is able to intercept all the messages. The Iolus [Mit97] implements a leave-sensitive rekeying policy.

Join-sensitive Rekeying Policy
Groups implementing a join-sensitive policy rekey only after membership `JOIN` events. The threat model implied by join sensitive groups states that any member joining the group should not have access to past content. A join-sensitive rekeying policy, by itself, does not ensure that members who left the group or were ejected from the group are not able to access current session content. The assumption is that past members can be trusted to not be interested in current content.

In practice, a join-sensitive rekeying policy is likely to be used in conjunction with a time-sensitive or a leave-sensitive rekeying policy to limit the duration over which past members can access current session content.

Membership-sensitive Rekeying Policy
Groups implementing a membership-sensitive policy rekey after every membership event. The threat model implied by membership sensitive groups states that any group member joining the group will not have access to past content, and that members who have left the group will not have access to current or future content. This policy is the combination of leave-sensitive and join-sensitive rekeying.

Applications with comprehensive security requirements will likely need a membership-sensitive rekeying policy. Providing strong guarantees for message delivery (e.g. atomicity, reliability) often requires tight controls over the message content. Without tight controls, the guarantees may be circumvented. The RAMPART [Rei94] system provides a type of membership-sensitive service.

Other Rekeying Policies
Applications may require a combination of the above policies or may use different policies depending on the application state and the specific attributes of an event.

In the business conferencing application, for example, the policy may be to rekey only when a member with the role Supplier leaves, but not when a member with the role Representative leaves. Allowing policies that make distinction between members may allow the application to optimize rekeying.

In certain circumstances it may be important for the group to be more sensitive at certain times, but less at others. Groups may wish sensitivity to be a function of group size or resource availability. In this way, a group can provide the strongest security model that is supported by the network and host infrastructure.

Time-sensitive rekeying can be useful in combination with any of the other schemes. Time-sensitive rekeying in combination with membership-sensitive rekeying, for example, helps ensure that an adversary cannot prevent session rekeying indefinitely without detection after a membership change event.

2.2 Application Message Policy

An application message policy states the types of security guarantees required for application messages. For example, an application wishing to ensure that no party external to the group is able to access content will define confidentiality as part of its policy.

The most common types of message security guarantees are: *integrity*, *confidentiality*, *group authenticity*, and *sender authenticity*. Confidentiality guarantees that no member outside the group may gain access to the session traffic. Integrity guarantees that any modification of an application message is detectable by receivers. A session key is typically used to obtain these guarantees. However, some guarantees such as sender authenticity are difficult to obtain without additional security infrastructure. Antigone supports these four message guarantees.

Note that a single policy need not apply to every message. In most applications, different messages will require different guarantees, depending on the nature of the message and the assumed threat model.

2.3 Membership Policy

Identification of the membership within a group session is an important requirement for a large class of applications. As evidenced by a number of group communication systems, achieving strong guarantees for the availability and correctness of group membership can be costly. In providing availability, any change in membership requires the distribution of the new group membership *view* to each member.

Conversely, members in another class of systems need not be aware of group membership at all. This is the model used in typical multicast applications. In this environment, providing other services (such as reliability, fault-tolerance, etc.) is commonly left to the application.

A membership policy indicates the availability of group membership information. If the policy states that group membership be supported, the group *view* (membership information) is distributed to each member as needed.

Antigone provides mechanisms to distribute keys with and without membership information, primarily to allow higher-performance applications when members do not need membership information.

Currently, Antigone does not attempt to guarantee the confidentiality of group membership. In general, hiding the group membership from members and non-members is difficult to do in current networks without introducing noise network traffic. This is primarily because the ability to intercept messages on the network allows access to the source and destination of packets (in case of unicasts) and at the multicast tree (in case of IP multicasts). In mounting this *traffic analysis attack*, an adversary may deduce a close approximation of group membership.

2.4 Process Failure Policy

A process failure policy states the set of failures to be detected and the security to be applied to the failure detection mechanism. The defining characteristic of a failure detection mechanism is the definition of its fault model. The fault model defines the types of behavior exhibited by a faulty process that the mechanism will detect. Typical crash models include fail-stop, message omissions, or timing errors [Mul93]. In the strongest (Byzantine failure) model, a faulty process may exhibit any behavior whatsoever.

A process failure policy may also state the need for secure failure detection. In securing the failure detection, the group may be protected from the masking of process failures by an adversary. However, protecting the group from an adversary who attempts to generate false failures may be more difficult. Failures may be forced by blocking all communication between the group participants. This is a denial of service attack, which is difficult to address in software.

Antigone supports detection of fail-stop faults of group members. To prevent problems due to timing errors, synchronized clocks and timestamps are not used in the Antigone protocols. However, some mechanisms for failure detection and time-sensitive rekeying in Antigone do rely on timeouts at individual processes. A process whose clock progresses at an incorrect rate may take longer to detect failures of other processes (if its clock progresses too slow) or may mistakenly assume that there is a failure (if its clock progresses too fast and thus times out).

3 Related Work

In this section, we review several of the key concepts and known techniques in the design of frameworks for group communication. Two fields applicable to our investigation are secure group communication and design approaches for configurable protocols. We present some of the major findings in each of these areas.

Much of the existing technology on which group communication is based was originally implemented in the ISIS [Bir93] and later HORUS [RBM96] group communication systems. Using these frameworks, developers can experiment with a number of protocol features. One important feature of the HORUS system is the introduction of a comprehensive security architecture. An element of this architecture is a highly fault-tolerant key distribution scheme. Process group semantics are used to facilitate secure communication. The session key distributed to members in HORUS is maintained for the entire session, thus providing no sensitivity to membership change events. However, the vulnerability to attacks from past or future members is limited. Application messages have sender authenticity and may be confidential.

Virtual networks provide developers with an abstraction for building applications designed for (logically) local network traffic, but executed across across physically larger networks. The Enclaves system [Gon96] extends this model to secure group communication. Enclaves provides a leave-sensitive group, distributing a new *group key* after each member leave. Also, it is implied that the group key should be changed periodically (time-sensitive). Enclaves supports membership information but is not dependent on it. Messages have confidentiality and through point-to-point communication, sender authenticity.

The RAMPART system [Rei94] provides secure communication in the presence of actively malicious processes and Byzantine failures. The system uses secure channels between two members of the protocol to provide authenticity. Protocols depend greatly on the *consensus* of processes to reach agreement on the course of action. A membership-sensitive group is built on a secure group membership protocol. The security context is not changed through shared session keys, but the secure distribution of new group views. Messages have sender-authentication and integrity.

A limitation of many secure group communication systems is that they do not scale to larger networks. In [Mit97], Mittra defines the *1 effects n* failure, where a single membership change event effects the entire group. The Iolus system [Mit97], attempts to address this limitation by introducing locally maintained subgroups. Each subgroup maintains its own session key, which is replaced after a membership change event in the subgroup. Therefore, the effect of a membership change is localized to the subgroup. Iolus provides a leave-sensitive group, and all application messages are encrypted with the group session key. No membership information is distributed in Iolus.

Key hierarchies [WHA98, WGL98] provide an efficient alternative to subgrouping in achieving scalable, secure groups. A key hierarchy is singly rooted *n-ary* tree of cryptographic keys. The interior node keys are assigned by a session leader, and each leaf node key is a secret key shared shared between the session leader and a single member. Once the group has been established, each member knows all the keys between their leaf node key and the root. As changes in membership occur, rekeying is performed by replacing only those keys known (required) by the leaving (joining) member. Rekeying without membership changes (as needed in time sensitive groups), can be achieved by replacing the root key. Thus, the total cost of rekeying in key hierarchies scales logarithmically.

The Group Key Management Protocol (GKMP) [HM97] attempts to minimize the costs associated with session key distribution by loosening the requirement that each session key be independent of others. After being accepted into the group, newly joined members receive a *Key Encrypting Key* (KEK) under which a future session key will be delivered. A limitation of this group is that misbehaving members can only be ejected by the establishment of a new group. GKMP reduces the costs of authentication by introducing a peer-to-peer review process in which potential members are authenticated by different members of the group. The joining member's authenticity is asserted by existing members. GKMP provides a time-sensitive group by rekeying at the end of a session key's *lifetime*. Note that this is a key management protocol, and as such does not mandate how the session key is used. No membership information is provided to members.

The IPSec [KA98] standards attempt to achieve Internet security by introducing security mechanisms at the network layer. The Scalable Multicast Key Distribution (SMKD) [Bal96] standard extends this approach to the multicast environment. Intended as an extension to Core Based multicast routing [BFC93], SMKD uses the router infrastructure to distribute session keys. As the multicast tree is constructed, leaf routers obtain the ability to authenticate and deliver session keys to joining members. Thus, the cost of authentication and session key delivery can be distributed among the leaf routers. Similar to GKMP, SMKD provides a time-sensitive group which is rekeyed at the end of a session key's lifetime.

With respect to security, the policies implemented by these systems are essentially fixed. Applications must select a solution that provides a policy that best fits a minimal set of requirements. However, the accepted solution may provide unnecessary functionality at an increased cost.

The *micro-protocol* [HP94, HS98] design methodology is useful when constructing systems with dynamic protocol stacks. Using micro-protocols, a system designer may decompose the facilities provided by protocols into their atomic

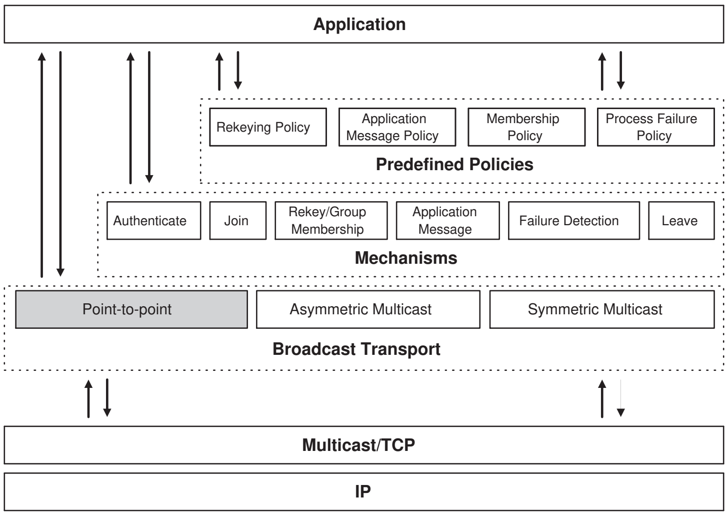

Figure 1: The architecture of Antigone consists of three layers; the broadcast transport layer, the mechanism layer, and the predefined policy layer. The broadcast transport component provides a single broadcast abstraction for environments with differing levels of support for multicast. The mechanism layer provides a set of primitives used to implement application policies. The predefined policies layer provides a suite of general purpose policies. Where required, applications may define other policies by accessing the mechanism and broadcast transport layer directly.

components. *Composite protocols* are constructed from a collection of the smaller micro-protocols. Differing facilities and guarantees may be provided through the composition of the micro-protocols. In [HS98], the authors define and demonstrate a suite of micro-protocols for maintaining group membership within a distributed application. Antigone uses the micro-protocol approach to achieve flexibility in implementing various security policies.

4 Architecture

Described in Fig. 1, the Antigone architecture consists of three software layers; the broadcast transport layer, the mechanism layer, and the predefined policy layer.

Though not typically associated with secure communication services, the broadcast transport component provides an abstraction for unreliable group communication. A reality of today's Internet is that there are varying levels of support for multicast services. Although significant effort has been expended on the development of WAN multicasting, no single solution has been found to meet the requirements of all users. Developers specify the level of multicast support of the target environment at run time. We describe the broadcast transport layer in Section 7.

The mechanism layer provides a set of mechanisms used to implement application policies. The mechanisms represent the basic features required for a secure group. Policies are flexibly defined and implemented through the selection and interaction of mechanisms. Associated with each mechanism is a micro-protocol [HS98]. The group protocol, called a *composite protocol*, is defined by the composition of the mechanism micro-protocols. We identify and describe the design of the Antigone mechanisms in Section 5.

The predefined policy layer provides a suite of general purpose policies. These policies represent those commonly provided by secure group communication systems. Clearly, there are many policies beyond what is available in this layer. Where required, an application may extend or replace these policies by accessing directly the broadcast transport and mechanisms layers. We describe how the predefined polices are implemented through the Antigone mechanisms in Section 6.

Authenticate

1. $A \rightarrow SL : A, I_0$ (*authentication request*)
2. $SL \rightarrow TTP : SL, A, I_1$ (*pair key request*)
3. $TTP \rightarrow SL : \{[\pi_{SL,A} = \{A\}_{K_{SL}} \oplus \{SL\}_{K_A}], I_1\}_{K_{SL}}$ (*pair key response*)
4. $SL \rightarrow A : SL, A, \{g, A, I_0, I_2, [\textit{policy block}], Pu_G\}_{\sigma_{SL,A}}$ (*authentication response*)

Join

5. $A \rightarrow SL : A, \{A, I_2\}_{\sigma_{SL,A}}$ (*join request*)

Rekey/Group Membership

6a. $SL \rightarrow A : g, S_{SL}, (A, \{g, SK_g\}_{\sigma_{SL,A}})\{H(g, S_{SL}, (A, \{g, SK_g\}_{\sigma_{SL,A}}))\}_{SK_g}$ (*key distribution*)

6b. $SL \rightarrow A : g, S_{SL}, (A, \{g, SK_g\}_{\sigma_{SL,A}}), B, C, D, \ldots, \{H(g, S_{SL}, (A, \{g, SK_g\}_{\sigma_{SL,A}}), B, C, D, \ldots)\}_{SK_g}$ (*key/group membership distribution*)

6c. $SL \rightarrow group : g+1, S_{SL}, (A, \{g+1, SK_{g+1}\}_{\sigma_{SL,A}}), (B, \{g+1, SK_{g+1}\}_{\sigma_{SL,B}}), \ldots,$
$\{H(g, S_{SL}, (A, \{g+1, SK_{g+1}\}_{\sigma_{SL,A}}), \ldots)\}_{SK_{g+1}}$ (*session rekey*)

Application Messaging

7a. $A \rightarrow group : g, A, [message], \{H(g, A, message)\}_{SK_g}$ (*with integrity*)

7b. $A \rightarrow group : g, \{A, [message]\}_{SK_g}$ (*with confidentiality*)

7c. $A \rightarrow group : g, \{A, [message]\}_{SK_g}, \{H(g, \{A, [message]\}_{SK_g})\}_{SK_g}$ (*with integrity and confidentiality*)

7d. $A \rightarrow group : g, A, [message], H(g, A, message)_{C_A}$ (*with sender authenticity*)

Failure Detection

8. $A \rightarrow SL : g, A, S_A, H(g, S_A)_{\sigma_{SL,A}}$ (*member heartbeat*)
9. $SL \rightarrow group : g, SL, S_{SL}, H(g, S_{SL})_{Pr_G}$ (*session leader heartbeat*)
10. $A \rightarrow SL : g, A$ (*key retransmit message*)

Leave

11. $A \rightarrow SL : A, \{g, A, \{g, B\}_{SK_g}\}_{\sigma_{SL,A}}$ (*leave request*)

Figure 2: Antigone Micro-Protocol Description - micro-protocols for the various operating modes. A *composite protocol* is constructed from the selection of a subset of these modes. In some configurations, some micro-protocols may be omitted entirely.

5 Mechanisms

We chose a composite protocol approach [HS98] for the design of Antigone mechanisms. In composite protocols, features are partitioned into modules consisting of shared state, messages, events, and micro-protocols. This approach has several desirable properties that make it applicable to Antigone. First, the modular design allows for the integration of future enhancements with minimal effect on other modules. Second, the micro-protocol approach allows state sharing between modules. We avoid the costs typically associated with state sharing between protocols (message headers).

Antigone provides mechanisms for providing the following functions; authentication, member join, session key and group membership distribution, application messaging, failure detection, and member leave. The Antigone micro-protocol variants associated with each mechanism are defined in Fig. 2. Before describing the micro-protocols, we explain the principals in the protocols and the notation used.

Fig. 3 shows the principals in an Antigone logical group. A distinct member of the group whose identity is known in advance to all members, called the *session leader* (SL), is the arbiter of group operations such as group joins, leaves, etc. We chose an arbitrated group because of its low cost and its applicability to many applications. For example, in a secure pay-per-view video broadcast application, the cable company would provide a session leader that enforces the desired access control and key distribution policy.

A *Trusted Third Party* (TTP) provides the mechanism for the session leader to authenticate joining *group members*. Each potential member, A, of a group (including the session leader) has a shared secret K_A registered with the trusted third party (TTP). This secret key is generated and registered with the *TTP* before the party attempts to join any session. We assume an out of band method for registering these keys and for informing everyone about the identity of the session leader.

In our protocol descriptions, we use the term *SL* to refer to the identity of the session leader, A to refer to a current or potential member of the session, and *TTP* to refer to the trusted third party. $\{X\}_k$ refers to message *X* encrypted under the key *k*. The view identifier, g, is used to uniquely tag the changing views of group membership. The term SK_g refers to a session key in view g, and SK_{g+1} for the (next) view $g + 1$. The term *I*, possibly with a subscript, refers to a nonce value. Key distribution protocols based on Leighton-Micali define a term $\pi_{A,B}$, called a *pair key*, used to support secret communication between collaborating members *A* and *B*. Derived from the pair key, the session leader and a potential member *A* maintain a shared secret key $\sigma_{SL,A}$. A cryptographic hash for the text x is described by $H(x)$. We use the MD5 hash algorithm [Riv92] in our implementation.

The format of the identity, nonce, key, and view identifier values used in our current protocol implementation is as follows. Each identity is a unique 16 byte null terminated ASCII string of alphanumeric characters. A potential member is assigned this value when registering a long term key with the *TTP*. Nonce values are unique 64 bit values. To ensure that nonces are not reused, some source of monotonic values, such as the system clock, may be used. Key format is algorithm dependent. The DES standard uses an eight byte key (including eight parity bits). In the future, as other ciphers are integrated into Antigone, we will need to support other formats. The view identifier g consists of a two parts. The first part is an eight byte, null terminated name string that identifies the group, used only for displaying and debugging purposes. The second is an eight bit nonce value. Each time a new view is created ($g + 1$), a new nonce is generated and appended to the group name string to create the new view identifier.

A $policy\ block$ is distributed by the session leader to each group member during the authentication process. Defined by the application, the policy block is an arbitrary byte string stating the group policy. We describe the use of this data by the predefined policies layer in Section 6.

The session leader creates an *asymmetric key pair* (Pu_G, Pr_G) during group initialization. The public key (Pu_G) is delivered to potential members during the authentication process. The public key is used to reduce the cost of sending secure heartbeats from the session leader.

Where sender authenticity is configured, we assume the existence of a certificate distribution service [HFPS98]. The certificate service would provide access to certificates for each group member (C_A). Note that no certificate distribution service is required in our protocol to generate or distribute the (Pu_G, Pr_G) asymmetric key pair.

We use DES [Nat77] for all encryption in the current implementation. Its inherent strength is evident from its 20-year history, yet its 56-bit key length has long been the subject of debate. Related algorithms such as triple-DES [ANS85] or DESX [KR96] offer the strength of DES with considerably longer keys. Our protocols are not tied to any specific property of DES, and may be replaced with other cryptographic algorithms as necessary.

We assume that all processes that have achieved membership, and thus have been authenticated, adhere to the system specification. We assume no member willingfully discloses its long term or session keys. All members trust the *TTP* not to disclose their long term key, and to generate pair keys according to the specification.

The following text describes each of the micro-protocol modules in Fig. 2. All cited message numbers refer to Fig. 2.

Authentication Mechanism - The authentication mechanism provides facilities for the authentication of potential group members. The purpose of this mechanism is twofold. First, the session leader authenticates the potential group member. Second, a shared secret between the two parties is negotiated. The shared secret, called the *shared secret key*, is later used to implement a secure channel between the two parties.

We use the Leighton-Micali key distribution algorithm [LM94] to authenticate the joining process and negotiate the shared secret. The main advantage of Leighton-Micali is low cost; it uses symmetric key encryption throughout, with none of the modular exponentiation operations associated with public key cryptosystems. Public key cryptography requires significantly more computation than symmetric algorithms. The de-facto standard for public-key cryptography, RSA, can be up to 100 times slower in software and 1000 times slower in hardware than DES, the predominant symmetric algorithm [Sch96].

A prospective member initiates the authentication process by sending a message to the session leader containing her identity and a nonce value (message 1). The session leader then obtains the pair key $\pi_{A,B}$ from the *TTP* (messages 2 and 3). Derived from two identities and their associated long term keys, the pair key is used to establish an ephemeral secure channel between the processes. To prevent replay attacks, the session leader verifies the encrypted nonce value I_1 included in the *TTP*'s response.

The session leader computes the shared secret key for communicating with *A* as follows. The session leader generates the value $\{A\}_{K_{SL}}$. This value is XOR-ed with the pair key $\pi_{SL,A}$ received from the *TTP*. The resulting value is a shared secret key ($\{SL\}_{K_A} = \sigma_{SL,A}$) that is used to create a secure channel between the session leader and the prospective member *A*.

Note that *A* need not communicate with the *TTP* to obtain the shared secret key $\{SL\}_{K_A} = \sigma_{SL,A}$; she can compute it directly. *A* decrypts the session key with this value and validates her nonce.

After obtaining the shared secret key, the session leader responds with an authentication response message (message 4). The response contains the identities of the session leader and the potential group member, and a block encrypted with the shared secret key $\sigma_{SL,A}$. The encrypted block contains the view (g) and group member (A) identifiers, the group member nonce (I_0), a session leader nonce (I_2), the policy block, and the group public key (Pu_G). Upon receiving this message, the receiver decrypts the contents and verifies the nonce I_0.

Join Mechanism - The join mechanism provides a member with facilities for gaining access to the group. The mechanism also provides measures to ensure the reliability of the join.

The potential group member (A) joins the group by transmitting message 5 to the session leader. Upon reception of message 5, the session leader validates the nonce value (I_2) passed to the joining member during the authentication process. If the nonce is not valid, the join request is ignored, and the group continues. If the nonce is valid, the new member is accepted into the group. The reaction of the session leader to the join request depends on the configured group model (see below).

Note that the join message is unforgeable and fresh. A session leader knows that a correct join message is fresh and was generated by the member because of the presence of the (I_2) nonce value encrypted under the shared secret key.

Mutual authentication is achieved through the verification of the secrets *A* and *SL* share with the *TTP*. The potential member must be in possession of the secret shared with the *TTP* to obtain the session leader nonce (I_2) used in joining the group in message 5. The session leader must be in possession of the secret shared with the *TTP* to determine the secret key shared with *A*. *A* is convinced that message 4 is fresh by validating the nonce value I_0 sent in the original request.

Rekey / Group Membership Mechanism - The Rekey/Group Membership mechanism provides for the distribution of group membership and session keys. We note the distinction between *session rekeying* and *session key distribution*. In session rekeying, all existing group members must receive a newly created session key. In session key distribution, the session leader transmits an existing session key.

The rekey and session key and distribution messages (6a, 6b, and 6c) all contain a group identifier (g), the latest session leader sequence number (S_{SL}), and a *Message Authentication Code* (MAC) calculated over the entire message, $H(\ldots)$. The group identifier and sequence number identify the current group context. The MAC ensures integrity of the message.

Session keys are distributed via *session key blocks* ($A, \{g, SK_g\}_{\sigma_{SL,A}}$). The intended member of each block is identified by the group member identifier (A). The remainder of the block is encrypted using the shared secret key ($\sigma_{SL,A}$). If the group identifier is decrypted by the receiver correctly (matches the identifier in the message header), the member is assured that the block was created by the session leader.

Message 6a contains a session key block for one member, and no group membership information. Message 6b contains a session key block for one member and enumerates the current group membership ($B, C, D, \ldots$). In message 6c, a session key block is generated for each member in the group. Group membership in message 6c is extracted from the session key blocks.

Rekeying is performed by the transmission of message 6c. Rekeying in Antigone is similar to key distribution after a `LEAVE` operation in the Iolus system [Mit97]. The session leader caches the shared secret keys, so creating this

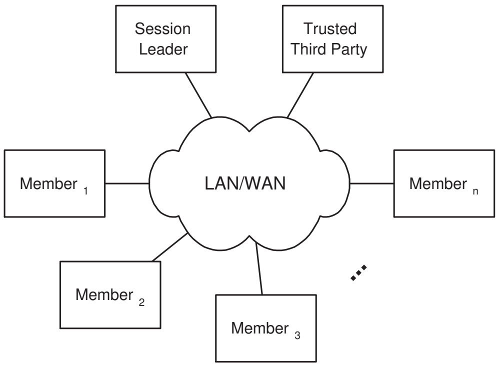

Figure 3: An Antigone group consists of an arbiter called the *session leader*, a service providing member authenticity called the *trusted third party*, and the group members. No assumptions are made about the network topology or connectivity.

message is fast: encryption of 24 bytes (8 bytes of new session key plus 16 bytes of new group identifier) per member. Using the cached, shared secret key, the receivers of this message extract the session key out of the session key block and begin using it immediately. The size of this message grows linearly with group size, and is potentially large. In systems that provide session key independence, keying material needs to be sent to each member individually. Therefore, the size of the message is large by its nature, not as a side effect of our design. Schemes that distribute a key to each member individually will transmit the same amount of data over many more messages.

A potential problem occurs during the transition of group views. During the rekeying process, application data such as continuous media may continue to be broadcast using a previous session key. Because of delays in the delivery of the session key, a process may receive a message encrypted with a session key that it does not yet or will never possess. An application may address this issue by buffering, double encryption, dropping packets, or other approaches. We present a detailed discussion the positive and negative aspects of several of these approaches in [MHP98].

We are in the initial stages of implementing a key hierarchy based rekeying mechanism (see Section 3). It has been shown that the costs of rekeying may be vastly reduced using this approach. In the future, we will study the effect of reduced rekeying costs on policy.

Application Messaging Mechanism - The application messaging mechanism provides for the transmission of application level traffic. Each application level message is secured using cryptographic keys distributed by the rekey/group membership mechanism, or through the use of external public key certificates.

The format of application messages is dependent on the type of messaging policy. We achieve message integrity through *Message Authentication Codes* (MAC) [Sch96], and confidentiality by encrypting under the session key. Message 7a shows the format of a message with integrity only, message 7b shows confidentiality only, and 7c shows a message with both integrity and confidentiality.

A MAC is generated by encrypting an MD5 hash [Riv92] of the message data under the session key. A receiver confirms the MAC by decrypting and verifying the MD5 hash value. If the hash is correct, the receiver is assured that message has not been modified by some entity external to the group.

Sender authenticity (message 7d) is achieved by digital signature [DH76]. The signature is generated using the private key exponent associated with the sender's certificate. Receivers obtain the sender's certificate and verify the signature using the associated public key. Note that a byproduct of the use of digital signatures is message integrity. Our current implementation does not support sender authenticity.

Failure Detection Mechanism - An application's threat model may require the system to tolerate attacks in which an adversary prevents delivery of rekeying messages from the session leader to the entire group or to a subset of members. In such a case, some members will continue to use an old session key, a security risk if the old key is compromised. Also, for accurate membership information, it may be necessary for the session leader to be able to detect fail-stop failures of members.

In Antigone, we provide *secure heartbeat* messages as the mechanism to detect failed processes. The session leader detects failed processes through member heartbeats (message 8). When some number of member heartbeat messages are not received by the session leader, the member is assumed failed and expelled from the group.

Group members confirm that the session leader is still operating by receiving the session leader's secure heartbeat messages (message 9). When some number (the value of which is a policy issue for higher layers) of session leader heartbeat messages are not received, the member can assume that the session leader has failed.

Heartbeat messages serve a dual purpose. In addition to failure detection, members use the heartbeats to ensure that they have the most current group state. The session leader's sequence number ensures that the heartbeat is fresh. The presence of the group identifier allows a group member to be certain that they are using the most recent session key. The heartbeats are encrypted to ensure that an adversary cannot fake heartbeats. Without these protections, an adversary may be able to prevent delivery of new session keys and trick members into continuing to transmit under an old session key indefinitely.

A member who fails to receive a current session key or membership information can attempt to recover by sending a key retransmit message (message 10). The key retransmit message indicates to the session leader that the member wishes to get the most recent group state. The session leader will send the most key/group membership distribution (message 6a, 6b, or 6c) in response to the key retransmit message. In this case, the process will be able to recover by installing the most recent session key and/or group membership.

It is worth noting that congestion that causes rekey or heartbeat message loss may be exacerbated by the resulting requests for more recent group state. This problem, known as *sender implosion*, is likely to limit the efficiency of Antigone in large groups or on lossy networks. We plan to introduce a retransmit mechanism similar to SRM [FJL$^+$97] to address this limitation. Using this approach, we distribute the cost of retransmission among the group by allowing any member to respond to requests for lost messages. Retransmission requests are made a random time after the loss is detected (the random time is computed as a function of measured distance from the session leader). Members noting a retransmission request suppress local requests, and wait until a retransmission is received or a timeout occurs. Conversely, members receiving retransmission requests for data (heartbeat or key messages) they have received wait a random time (which is a function of the distance from the requester) before retransmitting. If it is noted that some other member has performed the retransmission, the request is ignored. Note that this approach in no way affects the security of Antigone, but only serves to reduce the cost of retransmission request processing.

The Antigone infrastructure's goal at this level only provides mechanisms for reliable detection of session leader's failure and not recovery from its failure. Mechanisms for detection of failures can, if desired, be used to implement recovery algorithms using primary backup or replicated approaches at higher levels.

Leave Mechanism - This mechanism provides an interface for group members to gracefully exit the group. A member sends message 11 to indicate that it is exiting the group.

The leave mechanism has a secondary purpose. Using the micro-protocol defined in Fig. 2, a group member may request the ejection of other members from the group. To request an ejection, the requester places the identity of the member in the $\{g, B\}_{SK_g}$ block (as B). The session leader receiving a message with this format will eject the member in accordance with some local policy.

6 Policy Implementation

In this section, we show how flexible application policies may be implemented through the Antigone mechanisms.

Membership Awareness Policy

If members should not explicitly be given the list of current members in a group, then rekeying can be done via point-to-point messages to each member using message 6a. Otherwise, using message 6c via a multicast to rekey a group is generally more efficient. In either case, as pointed out in Section 5, we do not guarantee confidentiality of group membership against adversaries who are able to monitor network traffic and analyze packet flows in the group.

When a member fails to receive a rekey message, it can request a re-broadcast by sending message 10. If membership

Policy	JOIN	LEAVE	FAILURE	EJECT
Time Sensitive	N	N	N	N
Leave Sensitive	N	Y	Y	Y
Join Sensitive	Y	N	N	Y
Membership Sensitive	Y	Y	Y	Y

Table 1: The predefined rekeying policies may be defined by their sensitivity to membership events. Note that join sensitive groups are MEMBER_EJECT sensitive to allow for member ejection.

awareness is required in the group, the session leader can use message 6b or 6c to update the member; otherwise, the session leader can use the smaller message 6a to rekey the member, which is more efficient to encrypt and send.

Rekeying Policy Implementations

To implement a time-sensitive group, the session leader simply creates a new group identifier periodically, followed by group rekeying as describe above.

To implement a join-sensitive group, after a member joins (message 5), the session leader rekeys the session using either message 6a to each member or via message 6c.

If the policy is not join-sensitive, when a member joins, no rekeying is necessary. The new member is sent message 6b or 6a, depending on whether it needs to be provided the group membership list or not, respectively.

To implement a leave-sensitive group, when any member leaves (message 11), is ejected (message 11), or fails (detected via secure heartbeats), the session leader rekeys the group. The session leader rekeys the session by sending message 6a to each member or by multicasting message 6c to the group, depending on the membership awareness policy.

To implement a membership-sensitive group, session rekeying is done after joins, leaves, failures and ejections.

Application developers can implement other rekeying policies, given the mechanisms in the previous section. For example, the session leader can rekey only when certain members join or leave member in the group, or do time-sensitive rekeying in combination with membership-sensitive rekeying.

Application Message Policy Implementations

Several choices with different guarantees are available for sending messages (choices 7a, 7b, 7c, and 7d). It is up to the application to make judicious use of these available choices, depending on the requirements. We will discuss the performance implications of application message policies in the Performance section.

Predefined Policies

To simplify application development, Antigone provides a simple specification interface to select a group security policy. This specification interface allows selection from several common policies. Applications that require custom policies can, of course, implement their own policies directly though the Antigone mechanisms.

The predefined-policy layer in Fig. 1 uses the policy block in message 4 (see Fig 2) to send six fields (GG, SG, RK, HB, MM, and FP) to select a policy from a range of common policies.

The GG parameter is used to select from four implemented policies. The policies (TIME_SENS, LEAVE_SENS, JOIN_SENS, and MEMBER_SENS) correspond to the definitions presented in Section 2. Table 1 describes these policies in terms of their sensitivity to membership events.

The RK parameter is used to specify the rekey timer value. Each member resets the timer to the specified value when a new key is received. If the timer expires, the member considers the current session key to have expired and requests that the session leader send a new session key. The session leader should normally rekey a session prior to the expiration of this timer. Note that the combination of GG and RK parameters allow specification of rekeying policies that are both membership-sensitive as well as time-sensitive.

The SG parameter is used to specify the security guarantee on application messages. The guarantees that can be specified are: *confidentiality*, *integrity*, and *sender authenticity*. A side effect of the selection of any of these guarantees is that all application messages will have the *group authenticity* property. The group authenticity property states that messages can be verified to determine if they emanated from a member of the group. We outline the performance issues relating to these policies in Section 8.

The MM parameter is used to indicate if membership awareness is required by group members. If membership information is to be supported, it is distributed during every rekeying operation via message 6c or, in case of retransmit requests, via message 6b. Otherwise, message 6a is used to rekey members.

The failure policy (FP parameter) indicates whether failures of the session leader and the members are to be detected. If the process failure policy states failures are to be detected, heartbeat messages (9,10) are transmitted periodically. The failure detection mechanism will detect failed members as defined in the previous section.

The HB parameter is used to specify the periodicity of heartbeat messages, if any.

In general, if an adversary can prevent messages from being delivered, periodic heartbeats from the session leader are important. Since the session leader's heartbeats carry the group identifier and a sequence number, they help ensure that a group can be forced to use an old session key only for a period implied by the heartbeat interval.

Heartbeats are useful even when a combination of membership-sensitive and time-sensitive rekeying is used. For large groups, rekeying (message 6c) can be more expensive for the session leader compared to sending secure heartbeats (message 9). In such cases, the session leader can set the heartbeat interval to be lower than the rekey interval. The lower heartbeat interval ensures that members do not use an old key beyond that interval. In the absence of heartbeats, the rekey interval will define the bound for which an old key may be used.

7 Broadcast Transport Layer

Antigone provides three broadcast transport modes; *symmetric multicast*, *point-to-point*, and *asymmetric multicast*. In providing a single transport abstraction, Antigone supports network environments with varying levels of support for multicast.

The symmetric multicast mode uses multicast to deliver all messages. Applications using this mode assume complete, bi-directional communication is available via multicast. In effect, there is no logical difference between this mode and direct multicast.

The point-to-point mode provides a broadcast service based solely on point-to-point communication. All message traffic intended for the group is unicast to the session leader, and relayed to each group member via UDP/IP. However, as each message must be transmitted to group members individually, bandwidth costs increase linearly with group size. In some applications, these costs may be prohibitive. For example, a group of even modest sizes would have difficultly in maintaining a video transmission with reasonable frame rates.

In [AAC^{+}99], we describe our experiences with the deployment of a video application based on an early version of Antigone. The deployed system was to securely transmit video and audio of the September 1998 Internet 2 Member Meeting. The receivers (group members) were distributed at several institutions across the United States. While some of the receivers were able to gain access to the video stream, others were not. It was determined that the network could deliver multicast packets towards the receivers (group members), but multicast traffic in the reverse direction was not consistently available (towards the session leader). The problem was attributed to limitation in reverse routing of multicast packets. We present significant technical detail of this issue in [AAC^{+}99].

These problems coupled with the costs associated with a completely point-to-point solution lead us to introduce *asymmetric multicast*. This mode allows for messages emanating from the session leader to be multicast, and member messages to be delivered point-to-point. Members wishing to transmit a message to the group send the message to the session leader as a unicast. The session leader acts as a relay for all group communication. Any message intended for the group received by the session leader is re-transmitted via multicast. Thus, we reduce the costs associated with group delivery of a point-to-point solution to a unicast followed by a multicast.

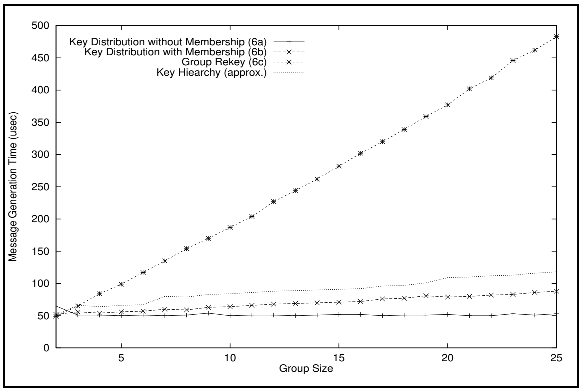

Figure 4: Group Membership/Session Key message generation costs - Measured and estimated costs associated with the message generation of groups of varying sizes and semantics.

Significant technical and administrative issues must be resolved before multicast services can be widely deployed in the Internet. In the interim, techniques such as asymmetric multicasting will be useful in the construction of group based systems.

The key advantage of a single abstraction is in its ability to handle updates in the networking environment. As symmetric multicast becomes universally available, neither Antigone nor applications built on it will require redesign to make use of the newly available service.

8 Performance

In this section we present a preliminary performance study of the Antigone framework. We illustrate the CPU costs of group key operations at the session leader and identify the throughput and latency characteristics of application messaging.

The experiments described in this section were performed on unloaded Intel 200MHz Pentium Pro workstations running FreeBSD kernel version 3.0. All tests were executed on an unloaded 100 MBit Ethernet LAN. Throughput and latency experiments included a single sender (which was the session leader) and nine receivers.

Fig. 4 shows the cost of group membership/session key message generation under varying group models and sizes. For comparison, we estimate the message generation costs associated with a key hierarchy approach.

Generation of the session key distribution message without group membership information (message 6a) should be constant under all group sizes. Creation of this message includes the generation of one session key distribution block and the generation of a *Message Authentication Code* (MAC). The figure indicates a constant cost of message generation.

The costs associated with key distribution with group membership message (6b) generation increases slightly with group size. This trend can be attributed to the increasing amount of data to be hashed. As the group membership grows, the costs associated with MAC generation increase.

The costs associated with the rekey message (6c) increases linearly with group size. A session key block is generated for each member, which requires a distinct cryptographic operation. Similar to the message 6b, the cost of MAC generation increases with group size. Because of the speed of the underlying cryptographic algorithms, increases in the cost of message generation due to MAC construction is significantly less than increases due to session key block construction.

The Key Hierarchy data in Figure 4 estimates the cost of rekey message with membership (similar to 6c) generation in a key hierarchy based group. In a key hierarchy, rekeying is performed by the distribution a number of keys which is roughly proportional to the log of the group size. Therefore, we see little increase in message cost as the group

Policy	Throughput	Latency
Integrity	2577 KB/sec	5710 usec
Confidentiality	1697 KB/sec	8698 usec
Integrity and Confidentiality	1577 KB/sec	9037 usec

Table 2: Application Messaging Performance - Maximum throughput by an application sending 1 KB messages and latency of end to end delivery for a 10 KB message.

becomes larger. This realization further indicates that a key hierarchy based rekeying mechanism will be a valuable extension to Antigone.

In Table 2, we show the throughput and latency characteristics of application messaging under varying security policies. Although not surprising, our experiments show that as the number of guarantees increase, so do the costs. Thus, policy has a direct affect on performance: stronger policies have less throughput and greater latency. There is a natural ordering to these guarantees. *INTEGRITY* (message 7a) is least expensive, *CONFIDENTIALITY* (7b) is more expensive, and the *INTEGRITY* and *CONFIDENTIALITY* (7c) is the most expensive.[2]

The ordering of application message generation costs mirrors the speed of the underlying cryptographic operations. We use DES [Nat77] to achieve confidentiality, and MD5 [Riv92] to achieve integrity. Our implementation uses the SSLeay v0.9.0b [HY98] `crypto` library. On the test machine, we found that DES encryption (≈ 3.7 Mbyte/second) is about $1/6$ the speed of MD5 hashing (≈ 21 Mbyte/second).

9 Conclusions and Future Work

In this paper, we presented the Antigone framework. Antigone provides a flexible interface for the definition and implementation of a wide range of secure group policies. Applications implement policy by composing and configuring secure group primitives called *mechanisms*. Thus, Antigone does not dictate the available policies, but provides facilities for building them.

The mechanisms provided by Antigone represent the set of features required to implement a secure group. The mechanisms include facilities for authentication, member joins, session key and group membership distribution, application messaging, failure detection, and member leaves. Policy-implementing software use these mechanisms to construct a feature set specific to the application context and the assumed threat model.

The target applications of Antigone require a low-cost solution. A result of this requirement is that the mechanisms in Antigone provide simple, but substantive, features for implementing various security policies.

To validate the primitives as well as to simplify development of applications, we have constructed a suite of general-purpose security policies. These predefined policies represent those that have been found useful or have been suggested as being useful by various group communication systems.

Though not typically associated with secure group communication services, Antigone provides an abstract interface for multicasting. A reality of the existing Internet fabric is its inconsistent support for multicast services. In deploying multicast-based solutions, we have found that though multicast connectivity in one direction is often possible, achieving bi-directional multicast is more difficult. As a result, we introduce a transport mode called *asymmetric multicasting*. In asymmetric multicasting, messages emanating from a single source use multicast, and all others use unicast. Antigone's implementation provides interfaces for symmetric multicast (bi-directional), asymmetric multicast, and point-to-point (unicast) group communication.

Our initial performance study indicates that as policy requirements increase, so do the performance costs. This is not a surprising result, but indicates the need for security infrastructures to support a range of security policies. In

[2] Currently, we have not implemented the sender authenticity guarantee (message 7d), and thus do not show its cost. As it requires private-key encryption, we expect sender authenticity to be the most costly guarantee to provide.

the near future, we will extend this study to include a more thorough analysis of latency, throughput, and scalability characteristics of Antigone groups within several networking environments.

An early version of Antigone has been integrated into the the `vic` Videoconferencing application [Net96]. A number of policy issues arose during the implementation and deployment of the resulting *Secure Distributed Virtual Conferencing* (SDVC) system [AAC+99]. The current design of Antigone represents the many refinements prompted by the analysis of SDVC.

Several challenges remain. Applications may have requirements for greater fault tolerance. The need for services that provide greater scalability is evident. In the future, we hope to investigate ways to meet such requirements, while retaining simple mechanisms that support flexible security requirements.

10 Availability

Source and documentation for the Antigone system are available from

`http://antigone.citi.umich.edu/`.

11 Acknowledgements

We would like to thank William A. (Andy) Adamson, Charles J. Antonelli, and Kevin Coffman for their assistance in the development of the Antigone system, Weston A. (Andy) Adamson for his help in constructing and maintaining the Antigone web-site, and the anonymous reviewers for their many helpful comments.

References

[AAC+99] A. Adamson, C.J. Antonelli, K.W. Coffman, P. D. McDaniel, and J. Rees. "Secure Distributed Virtual Conferencing". In *Communications and Multimedia Security (CMS '99)*, September 1999.

[ANS85] ANSI. "American National Standard for Financial Institution Key Management". *American Bankers Association*, 1985. ANSI X.917.

[Bal96] A. Ballardie. "Scalable Multicast Key Distribution". *Internet Engineering Task Force*, May 1996. RFC 1949.

[BFC93] T. Ballardie, P. Francis, and J. Crowcroft. "Core Based Trees (CBT)". In *Proceedings of ACM SIGCOMM '93*, pages 85–95. IEEE, September 1993.

[Bir93] K. Birman. "The Process Group Approach to Reliable Distributed Computing". *Communications of the ACM*, 36(12):37–53, December 1993.

[Dee89] S. Deering. "Host Extensions for IP Multicasting". *Internet Engineering Task Force*, August 1989. RFC 1112.

[DH76] W. Diffie and M.E. Hellman. "New Directions in Cryptography". *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[FJL+97] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. "A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing". *IEEE/ACM Transactions on Networking*, pages 784–803, December 1997.

[Gon96] L. Gong. "Enclaves: Enabling Secure Collaboration over the Internet". In *Proceedings of 6th USENIX UNIX Security Symposium*, pages 149–159. USENIX Association, July 1996.

[HFPS98] R. Housley, W. Ford, W. Polk, and D. Solo. "Internet X.509 Public Key Infrastructure, Certificate and CRL Profile". *Internet Engineering Task Force*, June 1998. (draft-ietf-pkix-ipki-part1-08.txt).

[HM97] H. Harney and C. Muckenhirn. "Group Key Management Protocol (GKMP) Architecture". *Internet Engineering Task Force*, July 1997. RFC 2094.

[HP94] N.C. Hutchinson and L.L. Peterson. "The x-Kernel: An Architecture for Implementing Network Protocols". *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1994.

[HS98] M. Hiltunen and R. Schlichting. "A Configurable Membership Service". *IEEE Transactions on Computers*, 47(5):573–586, May 1998.

[HY98] T. Hudson and E. Young. SSLeay and SSLapps FAQ, September 1998. http://psych.psy.uq.oz.au/ ftp/Crypto/.

[KA98] S. Kent and R. Atkinson. "Security Architecture for the Internet Protocol". *Internet Engineering Task Force*, November 1998. RFC 2401.

[KR96] J. Kilian and P. Rogaway. "How to Protect DES Against Exhaustive Key Search". In *Proceedings of Crypto '96*, pages 252–267, August 1996.

[LM94] T. Leighton and S. Micali. "Secret-key Agreement without Public-Key Cryptography". In *Proceedings of Crypto 93*, pages 456–479, August 1994.

[MHP98] P. McDaniel, P. Honeyman, and A. Prakash. "Lightweight Secure Group Communication". Technical Report 98-2, CITI, University of Michigan, April 1998.

[Mit97] S. Mittra. "Iolus: A Framework for Scalable Secure Multicasting". In *Proceedings of ACM SIGCOMM '97*, pages 277–278. ACM, September 1997.

[Mul93] Sape Mullender. *Distributed Systems*. Addison-Wesley, First edition, 1993.

[Nat77] National Bureau of Standards. "Data Encryption Standard". *Federal Information Processing Standards Publication*, 1977.

[Net96] Network Research Group, Lawrence Berkeley Laboratory. `vic` - Video Conferencing Tool, July 1996. http://www-nrg.ee.lbl.goc/vic/.

[RBM96] R. Van Renesse, K. Birman, and S. Maffeis. "Horus: A Flexible Group Communication System". *Communications of the ACM*, 39(4):76–83, April 1996.

[Rei94] M. Reiter. "Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart". In *Proceedings of 2nd ACM Conference on Computer and Communications Security*, pages 68–80. ACM, November 1994.

[Riv92] R. Rivest. "The MD5 Message Digest Algorithm". *Internet Engineering Task Force*, April 1992. RFC 1321.

[Sch96] Bruce Schneier. *"Applied Cryptography"*. John Wiley & Sons, Inc., second edition, 1996.

[WGL98] C. K. Wong, M. Gouda, and S. S. Lam. "Secure Group Communication Using Key Graphs". In *Proceedings of ACM SIGCOMM '98*. ACM, September 1998.

[WHA98] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key Management for Multicast: Issues and Architectures *draft-wallner-key-arch-01.txt* (Draft). *Internet Engineering Task Force*, September 1998.