Iolus: A Framework for Scalable Secure Multicasting

Suvo Mittra

Computer Science Department
Stanford University

suvo@cs.stanford.edu

Abstract

As multicast applications are deployed for mainstream use, the need to secure multicast communications will become critical. Multicast, however, does not fit the point-to-point model of most network security protocols which were designed with unicast communications in mind. As we will show, securing multicast (or group) communications is fundamentally different from securing unicast (or paired) communications. In turn, these differences can result in scalability problems for many typical applications.

In this paper, we examine and model the differences between unicast and multicast security and then propose Iolus: a novel framework for scalable secure multicasting. Protocols based on Iolus can be used to achieve a variety of security objectives and may be used either to directly secure multicast communications or to provide a separate group key management service to other "security-aware" applications. We describe the architecture and operation of Iolus in detail and also describe our experience with a protocol based on the Iolus framework.

1 Introduction

Multicast [7, 8, 9] is an internetwork service that provides efficient delivery of data from a source to multiple recipients. It reduces sender transmission overhead, network bandwidth requirements, and the latency observed by receivers. This makes multicast an ideal technology for communication among a large group of principals.

Although multicast has been very successful at providing an efficient, best-effort data delivery service to large groups, it has proven much more difficult to extend other features to multicast in a scalable manner. For example, scalable reliability, flow control, and congestion control all continue to be areas of very active research in the context of multicast, and the solutions that have been proposed are not as mature as their unicast counterparts.

Appears in *Proceedings of the ACM SIGCOMM '97*, September 14-18, 1997, Cannes, France.

Separately, the need to secure electronic information has become increasingly apparent. As multicast applications are deployed for mainstream use the need to secure multicast communications will become critical.

Moreover, as compared to unicast, multicast is inherently more susceptible to attack [2]. In the first place, multicast presents many more opportunities for interception of traffic. Second, when an attack does take place, multicast ensures that a larger number of principals is affected. Additionally, since multicasts are generally well advertised and the multicast addresses well-known, it becomes easier for an attacker to target an attack. Lastly, multicasts typically involve a "crowd" of principals and this can potentially make it easier for an attacker to pose as another (legitimate) principal or to try attacks in parallel.

Unfortunately, while the protection of unicast communications is well understood, multicast security has received scant attention. The few existing proposals for securing multicast communications neither address the unique requirements that arise from the group communications model of multicast nor have key management schemes that can scale to large groups [14] or groups with highly dynamic memberships.

In this paper, we examine and model the differences between securing multicast (or group)[1] communications and securing unicast (or paired) communications. We argue that they are fundamentally different in terms of key management. Furthermore, we show how these differences create scalability problems for many typical applications.

We then propose Iolus: a novel framework for scalable secure multicasting based on the notion of a *secure distribution tree*[2]. The Iolus framework is very general and can be used to achieve a variety of security objectives. At an abstract level, the Iolus framework enables scalable, secure communication among a group of principals using multicast. How this capability is utilized is left to the protocol:

1. Iolus can, of course, be used to build protocols that *directly* secure multicast communications. When used

[1] What we refer to as a multicast security protocol may be more abstractly termed a group communications security protocol. We use the term multicast security protocol because, in practice, such a protocol would typically be used to protect multicast traffic.

[2] In Greek mythology, Iolus helped Hercules with the first of his twelve labors — slaying the many-headed water serpent Hydra.

in this way, Iolus-based protocols can secure arbitrary multicast transmissions.

2. By using the secure multicast communications capability as a base, Iolus can also be used to build protocols that provide an independent group key management service for other "security-aware" multicast applications. For example, in the latter mode, an Iolus-based protocol could be used to implement common key management service that works in conjunction with other specialized applications (e.g., for audio and video) which together make up a complete, component-based multicast solution.

3. The need for group key management is not limited to multicast applications. As an example, consider a database that can be queried (using unicast) by a (changing) set of principals that logon and logoff the database. In effect, the current set of authorized principals constitutes a group. Instead of verifying the authorization for each principal separately on a per-query basis, the authorization for any query could be verified based on the possession of a common group key. The advantage of using a common group key is that it obviates the need to separately retrieve and apply the principal's keying material on each query. Iolus can also be used to build protocols that provide a group key management service to unicast applications.

Next, we describe the architecture and operation of Iolus in detail. As an example of its application, we have implemented a user-level software package that uses a protocol based on Iolus framework. This software has been used for secure multicast data delivery and has also been used with the `vic` video-conferencing application as a separate keying service. We describe our implementation experience and provide preliminary performance results.

The remainder of this paper is organized as follows. The next section examines the fundamental issues in secure multicast and the differences between secure multicast and secure unicast. Section 3 explains how these issues lead to scalability problems. Section 4 presents the requirements and constraints that shaped the design of the Iolus framework. Section 5 describes the architecture of the Iolus framework. Section 6 presents an operational overview of Iolus. Section 7 describes the use of Iolus as a key management service. Section 8 discusses deployment issues. Section 9 describes our implementation experience and some preliminary performance results. Section 10 presents related work. Finally, in Section 11 we conclude.

2 Network Security: Adding Multicast

The fundamental task of any network security protocol is to allow authorized principals to communicate securely over an insecure network where an attacker is assumed to be able to read, modify, or delete the raw communications. Typically, this is achieved by creating a security association between the authorized principals through authentication and key exchange. The security association defines a set of keying material shared only by the authorized principals that then can be used for a variety of security objectives such as authentication, confidentiality, and integrity. While the concept of security associations is well understood in the context of unicast, the multipoint-to-multipoint model of multicast inherently changes the meaning and management of security associations.

Consider what happens under unicast. Two principals decide to communicate and employ a (unicast) network security protocol to setup a security association between them. This association then allows the pair to communicate securely. Note that the security association is entirely static — it begins when the two principals begin communication and is destroyed when they end their communication.

Something similar occurs under multicast, but instead of two principals forming a pair, any number of principals form a group. And, whereas the security association is static in the unicast case, beginning and ending with the existence of the pairing, the security association must be dynamic in the case of multicast changing as the membership in the group changes.

Thus, a secure multicast can be thought of as consisting of blocks of time (see Figure 1). A principal may be authorized to participate in the secure multicast during various blocks of time. A multicast security protocol must ensure that a principal is only allowed to participate during those periods when it is authorized to do so. Mapping to actual multicast operations, the demarcation between blocks corresponds to members joining and leaving the group. The security association and thus the group keying material (K_{GRP} in short) it defines must be changed on each join and leave. This change is to ensure that a joining entity is not able to access previously multicast data and a leaving entity is not able to continue to access data multicast after it leaves the group[3].

To map this concept to a simple real-life situation consider a video-conference between a group of prosecuting attorneys who are discussing strategy for a criminal trial. At various (possibly overlapping) times, they wish to interview certain other people (e.g., police officers, witnesses, the defendant, and his attorneys). These people need to participate in the secure multicast, but only while they are being interviewed — they should not have access to any other previous or future communications.

Note that we are not implying that the keying material must be changed on each and every join or leave. That is an application-dependent policy decision. However, a multicast security protocol must be *prepared* to change the keying material on each and every join or leave to protect the integrity of the current group.

In general, the design of a given network security protocol is a complex business which must take into consideration a large number of factors. However, the management of a dynamic security association and its keying material is the basic difference between unicast and multicast security protocols.

[3] Of course, another option is to make the group immutable and start a new group whenever membership needs to change, but that is extremely expensive and even less scalable because it requires re-authentication on the part of all the group members.

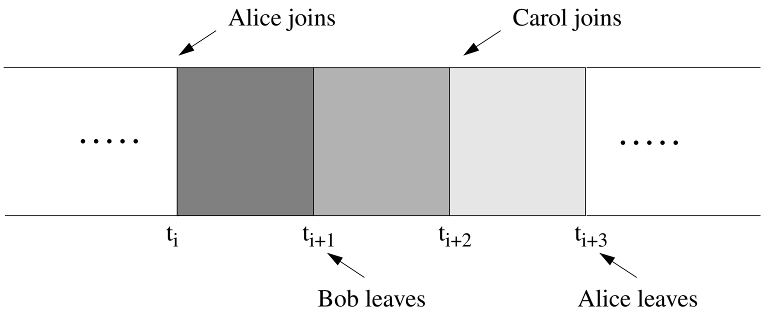


Figure 1: An Example Timeline for a Secure Multicast

3 The Scalability Problem

Loosely speaking, multicast protocols exhibit two types of scalability failures that are specific to multicast:

1. a **1 affects n** type failure which occurs when the protocol allows the action of one member to affect the entire group. For example, DVMRP [9] allows this to happen whenever a new sender joins the multicast group. A new source-based delivery tree rooted at the sender has to be built and all routers have to update state information.

2. a **1 does not equal n** failure which occurs when the protocol cannot deal with the group as a whole and instead, must consider the conflicting demands of each member on an individual basis. For example multicast flow control protocols exhibit this property. At any given time some receivers would like to adjust the transmission rate up, while others want to take it down.

In the last section we showed how unicast key management was fundamentally different from multicast key management. In this section, we will show how this difference leads to scalability problems for secure multicast protocols (of both the types just mentioned).

As noted previously, when a member joins the group, the group keying material (K_{GRP}) must be replaced and a new set of group keying material (K'_{GRP}) must be generated and distributed to the current group members as well as the joining member. There are a number of ways to do this. One simple method that works is to multicast K'_{GRP} to the current members using K_{GRP} to secure the transmission. Separately, the joining member can be apprised of the new key by unicasting K'_{GRP} to it using some unicast secure channel (e.g., the one used to request the join).

Thus, joins exhibit a **1 affects n** scalability failure because they require all members to process the change in K_{GRP} when one new member joins.

A member leaving the group, creates a much more difficult problem. As with joins a new set of keying material must be generated, but in this case it must be distributed to the remaining members in the group and there is no efficient means to communicate K'_{GRP} to them alone because there is no secret which is shared with them alone (and not also the leaving member)[4]. Indeed, K'_{GRP} is exactly this secret.

Given this chicken and egg situation, the keying material associated with each remaining member individually (i.e., unicast keys) must somehow be utilized to securely communicate K'_{GRP}. For example, using these unicast keys, a separate copy of K'_{GRP} could be sent to each of the remaining members[5]. The basic problem is that each of the members must be considered individually on each leave and this is extremely inefficient if the group is large or has a highly dynamic membership.

Thus, leaves suffer from both types of scalability failure; they result in a **1 affects n** scalability failure (in the same way as joins), but they also result in a **1 does not equal n** scalability failure.

Additionally, care must be taken to achieve at-least-once semantics when communicating updates to K_{GRP}, because members that do not receive the updates have the potential to create transient security breaches. That is:

- Receivers failing to receive a K_{GRP} update will not be able to continue decrypting communications and may also accept communications from members that have been removed from the group.

- Senders failing to receive a K_{GRP} update will continue to encrypt transmissions using an outdated K_{GRP} resulting in receivers that are unable to decrypt its transmissions. More importantly, security may be compromised because senders encrypting transmissions using an old K_{GRP} will allow former group members to continue decrypting transmissions.

If the underlying multicast service is unreliable, as it most often is, substantial additional processing will be required through the use of a reliable multicast protocol [12, 17]. And in many applications (e.g., multicasts of live audio/video), the use of such a protocol will only be for the purpose of keying updates thus adding to the complexity of the application.

Finally, these problems are exacerbated when the group has a highly dynamic membership, resulting in flurries of key update messages that increase the probability of transient security breaches and confusion.

[4] This assumes there has been no pre-distributon of keys. In any case, key pre-distribution serves only to transfer the problem to the time of a join rather than a leave.

[5] See Section 6.4 for other techniques and details.

Therefore, changes in group membership involve "heavy-duty" tasks not only from the standpoint of authentication and approval of the join/leave request (which in and of itself may be time consuming requiring database accesses and public key operations), but also from the distribution of new group keying material (especially on leaves).

These scalability problems are especially troublesome because many proposed applications of multicast have an ongoing need to maintain the integrity of the group for economic and other reasons. Of course, they also tend to have large group sizes and can often have highly dynamic group memberships.

For example, a large class of multicast applications are information dissemination services (e.g., data, audio, and videocasts). These applications can have large and dynamic groups especially when combined with the idea of micro-commerce and a "buy-on-the-fly" model for gaining access to content. Multi-player games and other interactive simulations also may have a need to ensure fairness among participants as they join and leave. Finally, consider group key management for services normally accessed using unicast. OAG, Inc., for example, has a service that provides continuously updated airline flight information. OAG charges for this popular service on a per-minute basis and so its group is large and constantly changing. Many other news and information retrieval services (e.g., LEXIS-NEXIS) have a similar business model and thus exhibit similar usage patterns.

4 Design Features and Requirements

We have discussed scalability as the driving force behind our framework. We now formally lay down all the design features and requirements that shaped our framework. Specifically, we wanted the framework to provide:

Scalability. As noted previously, a number of typical multicast applications have large groups and/or groups that have highly dynamic memberships. In addition, multicast is an emerging technology and all its possible applications in the future are not known. Therefore, the framework should not place any arbitrary limits on scaling.

Robustness. The framework should be able to adapt gracefully in the face of network disruptions (both malicious and accidental) and their effect should be minimized as far as possible. To that end, the framework should avoid the use of hard-state and rely as far as possible on soft-state. It should also attempt to localize the effects of disruption as far as possible, and try to avoid single points of failure.

Security Objective Independence. Network security protocols can be used to achieve a variety of security objectives. The framework is about enabling secure multicasting through scalable key management. As such, it should not define what specific security objectives can or cannot be met.

Security Technology Independence. Given a set of security objectives it usually possible to provide them using multiple cryptographic algorithms and protocols. The choice of a given algorithm or protocol is dictated by factors such as security threats, performance concerns, patent issues, and export limitations. Thus, the framework may assume the availability of standard cryptographic algorithms and protocols, but it should not stipulate the use of any specific algorithms or protocols.

Communications Protocol Independence. To be as widely applicable as possible, the framework should not make any discretionary assumptions concerning the services offered by the network or the communications protocol. It should be possible to implement the framework on any network that supports best-effort unicast and multicast services[6].

Protocol Layer Independence. Finally, there has been considerable discussion in the security community concerning the "right" layer in which to support security. That discussion is beyond the scope of this paper, but it should be possible to implement the framework at any layer that supports the basic communications services mentioned above.

5 The Iolus Framework

From our earlier discussion, it can be seen that large and/or dynamic groups do not scale up in terms of multicast security. Whenever the membership changes, all group members are affected. This creates an inherent scalability problem.

The Iolus framework seeks to directly address this problem by completely doing away with the idea of single flat secure multicast group. Instead, Iolus substitutes the notion of a *secure distribution tree*. The secure distribution tree is composed of a number of smaller secure multicast "subgroups" arranged in a hierarchy to create a single *virtual* secure multicast group (see Figure 2).

Scalability is achieved by having each subgroup be relatively independent. More specifically each subgroup in the secure distribution tree has its own multicast group (with its own address) and can be created using any suitable multicast routing protocol (e.g., DVMRP, CBT [1], PIM [10]). Moreover, each group has its own subgroup keying material (K_{SGRP} in short) and there is *no* global K_{GRP}. Thus, when a member joins or leaves, it joins or leaves only its local subgroup. As a result, only the local K_{SGRP} needs to be changed and the scalability problem is greatly mitigated.

Subgrouping for scalability is only the first component of our framework. The creation of a single secure multicast group image is the second. This is akin to the creation of a single system image in distributed systems. To do this, Iolus introduces two new types of entities that manage and connect the various subgroups. These are the *group security controller* which manages the top-level subgroup and the *group security intermediaries* (GSIs), one per subgroup, which manage each of the other subgroups. Generically they are called *group security agents* (GSAs).

Because of the single secure multicast group image, very little changes for senders and receivers under Iolus. They continue to send and receive as they ordinarily would. The

[6] Of course, it should be noted that multicast can be simulated using either unicast or broadcast, albeit with a loss of efficiency.

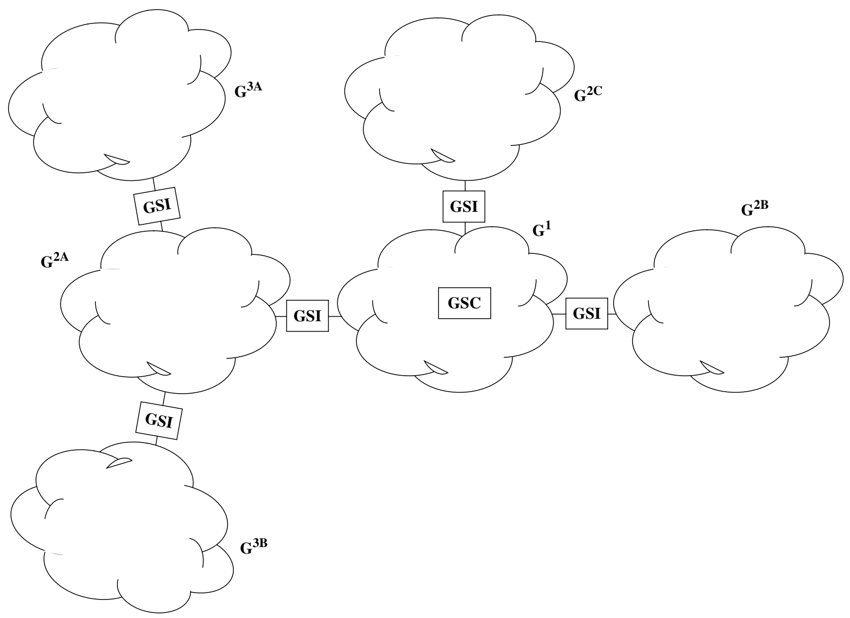


Figure 2: Example of a Secure Distribution Tree

only difference is that instead of all of them joining a single multicast group, they join separate local multicast groups[7]. The GSAs connect the subgroups and "conspire" to deliver locally multicast data to all the subgroups in the virtual group.

More specifically, the GSAs form a hierarchy of subgroups. The GSC maintains control of the top-level subgroup at the root of the secure distribution tree. It is ultimately responsible for the security of the entire group. GSIs are special trusted servers that are authorized to act as proxies of the GSC or their parent GSIs and control their local subgroup. The GSIs are grouped according to levels within the secure distribution tree. GSIs at a given level join the subgroups of the GSI at the next higher level or the subgroup of the GSC. They form a bridge between subgroups by receiving data multicast in their parent or child subgroups and re-multicasting to their child or parent subgroups respectively. Although it may seem that this would require decryption and re-encryption of the entire packet (to account for the different K_{SGRP}s in the two different subgroups) we will show that this is not the case.

Note that the hierarchy imposed on the subgroups serves a dual purpose. It nicely captures the delegation of authority between the GSC and the GSIs at different levels. But, it also simplifies how data is re-multicast throughout the different subgroups without incurring loopbacks.

Although there are a number of other deployment issues, we defer further discussion (see section 8 for details) and instead first present an operational overview of the framework.

Please note that in the remainder of the paper, we use the word *group* to refer to the virtual secure multicast group as a whole, while we use the term *subgroup* when we refer to a specific subgroup in that virtual group. Thus, a member joins the secure multicast *group* by accessing its local *subgroup*.

[7]This could be hidden by using a directory service (like sd) to automatically map a global multicast group to the local multicast subgroup (see Section 8.3 for details).

6 Iolus Operational Overview

We now provide a detailed description of the operation of the Iolus framework by considering each phase in the operation of a protocol based on it. The description in this section concerns the use of Iolus as a complete framework for securing multicast communications. In the next section, we describe an use of Iolus as a separate key management service.

Please note that that the description below is mainly from the point of view of the Iolus framework. Other operations may be required by the underlying communications protocol (e.g., core selection and bandwidth reservation) but these are not a part of the secure multicast protocol and are not discussed here.

6.1 Startup

Strictly speaking startup of the secure multicast group requires only that the GSC for the group be started. The GSC is supplied with an access control list (ACL) that it uses to set the security policy concerning who may have what access to the group. Once it is started, GSIs and other members apply to join its subgroup.

As a matter of policy some or all the GSIs may also be started at this time, but this is not necessary; they can also wait and see if they have any members interested in their subgroup first.

HDR	$\{K_{GRP}'\}_{K_{GRP}}$

Figure 3: GRP_KEY_UPDATE message on a JOIN

HDR	$\{K_{GRP}'\}_{K_{GSA\text{-}MBR_1}}$	$\{K_{GRP}'\}_{K_{GSA\text{-}MBR_2}}$	.....	$\{K_{GRP}'\}_{K_{GSA\text{-}MBR_n}}$

Figure 4: GRP_KEY_UPDATE message on a LEAVE

6.2 Joins

To join a secure multicast group, a sender or receiver locates its designated GSA and communicates a `JOIN` request to it using a *secure unicast channel*[8].

Upon receiving the `JOIN` request, the GSA checks its database and decides whether to approve or deny the request. Assuming the request is approved, it (1) generates a secret ($K_{GSA-MBR}$) to be shared only with the new member, (2) stores this secret along with any other relevant information concerning the new member in a private database it maintains, and then (3) communicates $K_{GSA-MBR}$ to the new member using the secure channel. The usage of $K_{GSA-MBR}$ as a way to isolate communications within a multicast will become clear shortly.

As described in section 3, the GSA now needs to change K_{SGRP} and make K'_{SGRP} known to the current members of the subgroup and the joining member. To do this, the GSA multicasts a `GRP_KEY_UPDATE` message containing K'_{SGRP} encrypted with K_{SGRP} to the current multicast subgroup (see Figure 3). It then communicates K'_{SGRP} to the joining member via the separate secure channel.

Of course, in order to process the `JOIN`, the GSA (if it is a GSI) must itself be a part of the secure multicast group. If it is not, then it needs to follow a similar procedure to join its parent subgroup. The only difference between the `JOIN` of a GSI as opposed to that of a sender or receiver is that the GSI is also supplied (via the secure channel) an ACL or other database that it can use to process `JOIN`s of its own.

6.3 Refreshes

Each `JOIN` has a expiration time associated with it. If the `JOIN` is set to expire and the member wishes to remain in the group it must send a `REFRESH` message using the secure channel. In this way the GSA only has soft-state associated with it. This allows the framework to gracefully handle network partitions. Subgroups that have become partitioned from the top-level subgroup are implicitly removed from the group after some threshold time has elapsed.

[8]Here, by secure channel we mean any unicast communications channel secured using a unicast security protocol. The exact protocol is not important; any unicast security protocol that provides mutual authentication with key exchange can be used.

6.4 Leaves

Leaves occur under two conditions: (1) a member wishes to voluntarily leave the subgroup in which case it sends a `LEAVE` request to the GSA, or (2) the GSA wants to expel a member of the subgroup and sends a notification to that effect to the expelled member.

Either way, K_{SGRP} needs to be changed to disallow the leaving member's continued participation in the subgroup. As described in section 3 there is no keying material that can be used to securely communicate K'_{SGRP} only to the remaining members of the subgroup.

As we mentioned before, one straightforward scheme for handling this problem is to send a copy of K'_{SGRP} to each member encrypted with that member's $K_{GSA-MBR}$. Note, however, that this does not imply the use of unicast. Instead of sending separate unicast messages to each member of the group, the GSA simply multicasts one message containing n copies of K'_{SGRP} (assuming n remaining members) each encrypted with a different member's $K_{GSA-MBR}$ (see Figure 4). In this way, a more efficient single message of size $O(n)$ can replace $O(n)$ separate messages[9].

Although we propose the use of the above scheme in Iolus, it should be understood that there are also numerous cryptographic techniques for handling this problem. For example, the literature describes methods that make use of group extensions to the Diffie-Hellman key exchange [5, 18, 24], secure locks based on the Chinese Remainder Theorem [6] or polynomial interpolation [13], and secret sharing schemes [4].

Unfortunately, given our purposes, these techniques are only of theoretical interest because they provide performance that is no better than the simple scheme presented above. To our knowledge, they all require computation that, at a minimum, is proportional to the number of remaining members (because they consider each member's keying material separately) and in addition the per-member computation is relatively expensive. Besides this, the resulting mathematical structure (i.e., the key) is sometimes quite large because its size is proportional to the number of members.

Finally, note that if the GSA is not the GSC and if after the leave the GSI is has no more children interested in the

[9]Sending one large message instead of sending multiple small messages is not a solution to the scalability problem. In a large scale multicast n is very large and having to send such a message on each change in membership is not scalable (especially since it must be sent reliably).

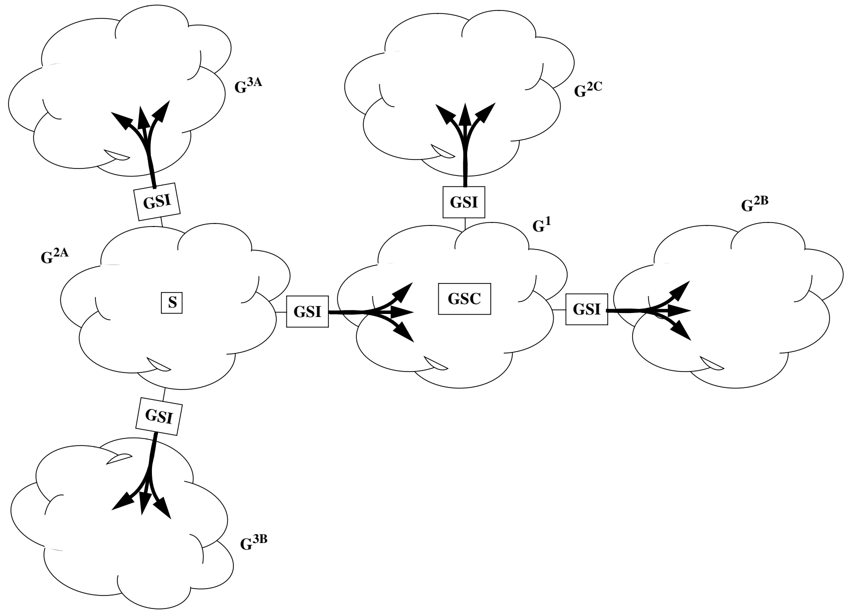


Figure 5: Data multicast in group G^{2A} is re-multicast throughout the secure distribution tree

secure multicast group, the GSI can in turn leave its parent's subgroup.

6.5 Data Transmission

Under Iolus, sending data is not as simple as multicasting the data to the group encrypted[10] with K_{SGRP} because such a multicast will only reach the local subgroup. Instead the hierarchy of multicasts must be considered and there must be some mechanism for the entire secure multicast group to receive the transmissions. We present two related methods.

The first method is simple and completely transparent to the sender. The sender multicasts the data directly to its local subgroup encrypted with K_{SGRP}. The parent GSI (if this is not the top-level subgroup) listens for multicast transmissions, decrypts them, and then re-multicasts them to its parent subgroup encrypted with the K_{SGRP} of its parent subgroup. In a similar manner, child GSIs of the subgroup receive multicast transmissions, decrypt them, and then re-multicast them to their child subgroups encrypted with the K_{SGRP} of their child subgroups. Since this process will repeat in the other subgroups where the data is re-multicast, the data will eventually reach every subgroup (see Figure 5).

Although it may seem that having the GSIs decrypt and re-encrypt data would require an enormous amount of encryption bandwidth, we can use the old trick of indirection to sharply reduce the cost of re-encryption. Instead of having the sender directly encrypt the data with K_{SGRP}, the sender generates a random key on a per-transmission basis and uses this key to encrypt the data. It then includes this key with the data, encrypted with K_{SGRP}. In this way, decrypting and re-encrypting the packet is reduced to simply decrypting and re-encrypting the random key.

While this technique is efficient and will suffice for most applications, it may result in *transient* security breaches of the type described in section 3. As noted in that section, the problem is due to the fact that senders and receivers may not have the latest K_{SGRP}. Thus senders may send data secured using an outdated K_{SGRP} allowing former group members to decrypt the data. At the same time receivers using an outdated K_{SGRP} may accept data from senders who are no longer in the secure multicast group.

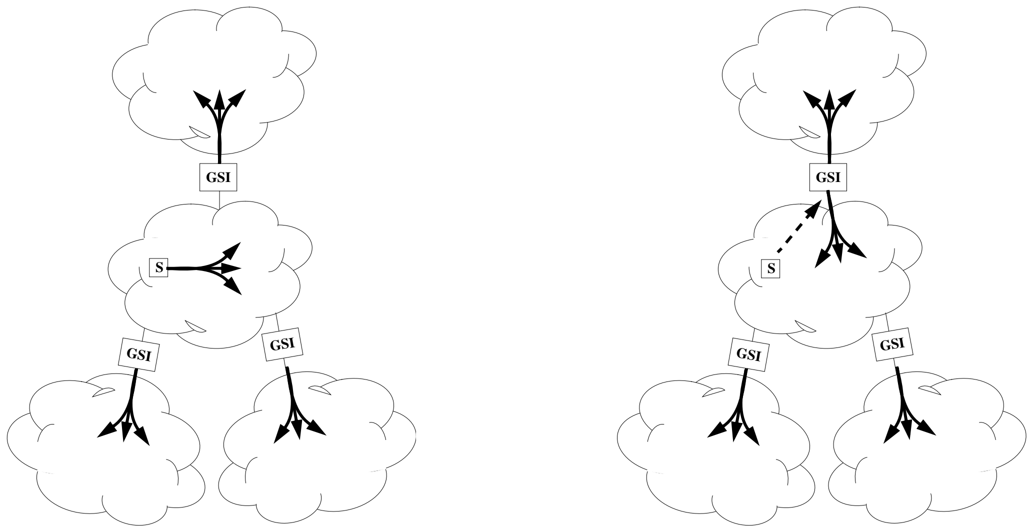
We present the solution as our second method. Instead of senders multicasting directly to the group, senders unicast the data to the GSA encrypted with their unique $K_{GSA-MBR}$. The GSA then decrypts the data, re-encrypts it with K_{SGRP}, signs it, and then multicasts it to the group as well as to its parent subgroup (if any). In this way, senders do not use K_{SGRP} obviating the possibility of their using an outdated K_{SGRP}, while receivers have the assurance of their GSA's signature as proof that the message is from a valid source. And, security breaches are avoided without using a reliable multicast protocol (at the cost of an extra unicast). Figures 6a and 6b demonstrate the difference between these two methods.

6.6 Re-keying

With use, K_{SGRP} will "wear out" and will need to be changed. If a simple key-update scheme is used, K'_{SGRP} can be made known to the group by multicasting it to the subgroup encrypted under the old K_{SGRP}.

However, this has the weakness that it sets up a chain of keying material and a compromise in one link of the chain results in a compromise of all the keying material following it in the chain. Instead, since the K_{SGRP} is always local to

[10]Although, for purposes of clarity, we have framed the discussion here in terms of encryption, it is easy to see how K_{SGRP} can also be used for meeting other security objectives such as authentication and message integrity. For example, K_{SGRP} could also be used to derive keys for digital signatures.



(a) Direct Multicasting

(b) GSA-Assisted Multicasting

Figure 6: Two Methods for Sending

a given subgroup, we can use the scheme used for changing K_{SGRP} on leaves to apprise the group members of K'_{SGRP} without creating a chain or running into scalability problems.

6.7 Shutdown

As far as Iolus is concerned, ending a secure multicast requires only that the GSC for the secure multicast group be shutdown after sending `GRP_END` notification to all the members in the top-level subgroup. In a similar manner, any GSAs attached to the top-level subgroup will then multicast the `GRP_END` message to their subgroups and then shutdown, and so on.

7 Iolus as a Key Management Service

Thus far we have described Iolus as a complete security framework for multicasting which applications can use to directly send and receive data. However, there are actually two separate problems: (1) key management, and (2) securing and transmitting data. In Iolus these two operations are linked because the secure multicast group exists only in a virtual sense. In actuality, each member only has access to its local multicast subgroup which has its own K_{SGRP}. Therefore, the GSAs must be involved in both the key management and data transmission to create the single secure multicast group image.

However, there are many "security-aware" applications that already know how best to secure and send their data. They simply need access to some common K_{GRP} which they can then use to achieve their security objectives. The `vic` video-conferencing software [20], for example, has this capability. In these cases, Iolus-based protocols can be used as key management services.

To use Iolus as a key management service, members join a secure multicast group as before by accessing a local subgroup. The only difference is that instead of directly using the secure multicast group to transmit "real" data, the secure multicast group is used to transport a second set of keying material (K_{GRP2}). K_{GRP2} is generated by a specially designated member and sent to all the members using an Iolus-based protocol. This allows all the secure multicast group members to share a secret (K_{GRP2}) that they can use as keying material to secure data multicast in an unrelated and independent multicast group.

To complete the scheme, K_{GRP2} must be changed whenever membership in any of the secure multicast subgroups change. However, actually doing this on each membership change could potentially create a scalability problem similar to that described for joins in section 3. Therefore, we provide an approximation by changing K_{GRP2} "frequently." While this does not provide enough security for all applications, it works well for a great majority. For example, consider the information dissemination service we described earlier. Since it charges on a per-minute basis, it only cares that the integrity of the group is maintained on a per-minute basis (not on each join and leave). This can be easily accomplished by changing K_{GRP2} every minute.

Moreover, using Iolus as a group key management service has a number of other benefits:

- It decouples key management and distribution from data transmission. This allows the use of the best existing multicast communications protocols without the limits based on subgrouping. Indeed, it also allows the use of Iolus with non-multicast applications that nonetheless require group keying.
- It also decouples key management and distribution from key usage. This decoupling can be useful if, for

example, the group members want to use a propriety encryption algorithm which they do not wish to share with the GSAs.

- It has been suggested, that in the future, multicast conferences might be implemented in a modular manner as a collection of specialized applications connected to a "conference bus" [20]. In such a paradigm, Iolus could effectively provide a common key management service to the other applications, thus obviating the need for them to individually implement key management.

8 Deployment Issues

Thus far we have glossed over many deployment issues related to the Iolus framework. For example, we have assumed use of the secure distribution tree and its component GSAs, but we have not considered how it is built or how it is managed.

Of course, these issues are beyond the scope of the framework and can only really be discussed in the context of an actual protocol implementation and the network on which the protocol is used. However, to gain some understanding of these issues, we briefly consider them in this section though we caution the reader that these issues require further research.

8.1 Secure Distribution Tree Management

The management of the secure distribution tree encompasses two issues: (1) who controls the tree along with it component GSAs and (2) whether the tree is fixed or dynamic within the network, i.e., is it built for a single secure multicast and then destroyed or it is reused for other multicasts.

One possibility, is that GSAs are owned and operated by the principal in charge of the multicast group and the tree is setup dynamically to be used solely for one particular secure multicast group. This is most secure because the principal in charge of the multicast retains control over the entire tree and because the GSAs are not shared among different multicasts.

At the other end of the spectrum it is possible to imagine a situation where GSAs are basically fixed within the network and are owned by trusted third-parties (e.g., network service providers). When a principal wishes to initiate a secure multicast, it "rents" a tree from the provider. Obviously, this is not as secure as the first scheme, but it is more practical.

More likely, both these possibilities will exist. There may be some principals (e.g., media companies) that hold so many secure multicasts that they may own and manage their own GSAs, while most other people will simply avail themselves of the services offered by third parties.

8.2 Secure Distribution Tree Construction

A related issue is the construction of the secure distribution tree through the placement of the GSAs in the network. This is an important issue because, in practice, any scalability offered by the secure distribution tree (and so by Iolus) is heavily dependent on how well the members are actually distributed among the various subgroups.

Unfortunately, it is impossible to find an optimal placement for GSAs in the network because doing so requires knowledge of future joins and leaves. However, in practice there are "natural" placements for GSAs that will generally lead to scalability. This results from the fact that most networks are inherently hierarchical. Consider the Internet. At the top of the hierarchy there are major nationwide backbones. These lead to regional networks, that in turn lead to ISP networks or directly to corporate and campus networks. Finally, these networks may be divided into subnets. Therefore, it is quite natural to place GSAs at the entrance to each of these networks (especially at lower levels). For example, GSIs could be placed on firewall machines at the entrance to campus networks or building networks. This also suggests that there will not be many GSAs between any given sender and receiver.

Another option that will definitely ensure scalability is to use "dynamic subgrouping." That is, when a specific GSA becomes overloaded, another GSA is started at the same level and within the same parent subgroup. The membership of the overloaded GSA is then split and reallocated between it and the new GSA.

8.3 Secure Distribution Tree Discovery

To begin accessing a secure multicast under Iolus, a prospective member must first locate its designated GSA and thereby locate the secure distribution tree. The interesting issue here is that GSA location requires mapping from a global name (i.e., the name of the secure multicast group) to a local one (i.e., the name of the GSA handling the local subgroup of that secure multicast group).

If GSAs are more or less fixed, locating a GSA is easy because there is always some set of well-known GSAs. However, if the GSAs are dynamic, then the solution must also take their dynamic nature into account. We suggest two possible solutions: (1) have GSAs notify some well-known, local directory service whenever they start or stop handling a secure multicast group so that the directory service knows where to direct queries, or (2) use either multicasting or anycasting [22] to locate the GSA.

Note that GSA location is equivalent to mapping a global secure multicast group to the local multicast subgroup's address because the GSA has the address of the local multicast subgroup and can pass it to the newly joined member after authorization.

8.4 Admission Control

We have noted that the Iolus framework is not tied to any particular layer in the protocol stack and may be implemented at any layer that offers basic best-effort unicast and multicast communications. We have also noted that job of a network security protocol is to secure communications over an insecure network where attackers are assumed to have more or less free rein. In particular, keeping attackers off the network is beyond the scope of a network security protocol and requires external network admission control.

However, it has also been argued that current multicast communications protocols make the job of an attacker very easy [2]. That is, an attacker simply has to declare itself a receiver to receive multicast communications. Not only does this make it easier for the attacker to access the communication, but it also facilitates certain types of denial of service attacks. For instance, if the multicast routing implements rendezvous routing (as in CBT or PIM), an attacker could begin mass subscription to the group, displace the rendezvous point, and thus potentially cause communications to be degraded.

To counter these threats, it has been suggested that any secure multicast protocol must be combined with the multicast routing protocol and make use of routers to perform admission control (perhaps using some form of security-enhanced IGMP). However, this scheme has its own drawback because routers are inherently a part of the (insecure) network[11].

Although the Iolus framework could also be combined with multicast routing protocols, it should be noted that it is not necessary to do so in order to counter these threats. Protocol designers worried about these threats can include hooks to routers which will provide "advisory" admission control. That is, routers will contact their designated GSA to seek permission prior to extending the group. In this way good routers that can be trusted will provide the requisite control, while rogue routers which could not be trusted in any case will continue to do as they please.

9 Implementation Experience and Performance Results

Ultimately, the merits of the Iolus framework can be understood only after wide-scale deployment. However, we wished to have a better understanding of some key issues concerning the framework. More specifically we were interested in the following issues:

1. Complexity: How difficult would it be to implement a protocol based on the Iolus framework?

2. Application Interfacing: How difficult would it be for applications to use an Iolus-based protocol, either for secure multicasting or for key management?

3. Performance: What performance penalty is incurred by GSAs in forwarding data between multicast subgroups?

To answer these questions we have implemented a simple software package that uses an Iolus-based protocol.

The package consists of the GSC and GSI applications along with a Iolus client application that is used to interact with the GSAs. Actual applications do not directly interact with the GSAs. Instead, they use a library to interface with the Iolus client. This allows for the use of the secure multicast functionality while at the same time minimizing modifications to the actual applications.

[11]Here, we are *not* implying that network layer security is bad idea. Network layer security can be implemented end-to-end. We are simply saying that it is unwise to trust arbitrary routers in the network.

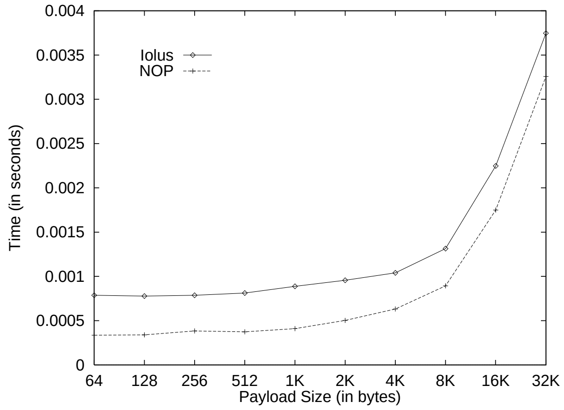


Figure 7: **GSA Multicast Forwarding Performance**

The client library is designed to be generic. That is, the library can be used by applications that want to directly secure multicast data, but it can also be used by applications that simply access to a group key management service. Thus, the library provides the standard `send()` and `receive()` functions along with a `getkey()` function for key retrieval. Besides these functions, the library contains a set of bootstrap routines to start up the Iolus client and begin interfacing to it using shared memory.

The cryptographic routines in the client library and in the other pieces of the package are implemented using the publicly available CryptoLib library [19]. CryptoLib includes most commonly used cryptographic primitives, e.g., DES, RSA, MD5, etc. The key reasons behind its choice were its general availability and portability. Our protocol utilizes DES [21] for encryption, keyed MD5 [23] for MAC computation, and Diffie-Hellman [11] with authenticated public values for mutual authentication with key exchange.

The basic package was coded in less than a month and contains a little more than 3400 lines of code. In particular, the client library was implemented in less than 250 lines of code and should not increase code size dramatically for most applications.

Because our library contains the requisite send and receive primitives, our package can be readily used to directly send and receive secure multicast data. However, we wanted to understand how difficult it would be to retrofit an existing "security-aware" multicast application to use our protocol as a key management service. To that end we decided to modify `vic` to use our protocol. `vic` is already "security-aware" in that it has the ability to encrypt and decrypt its transmissions given a key. Normally this key is supplied using `vic`'s GUI, but we modified `vic` to call our `getkey()` function prior to each send and receive. Besides this, the only other modification we had to make was to have `vic` call our bootstrap routines during startup. In sum, less than 30 lines of `vic` code in two files (i.e., `main.cc` and `net.cc`) were changed. Our experience with `vic` suggests that applications will find it easy to interface to our framework.

Finally, we wanted to gain some understanding of the performance aspects of our framework. In particular, our framework relies heavily on GSA forwarding of data between subgroups. To test the performance of multicast forwarding, we setup a test in which we measured the response time of the GSA on packets with payloads ranging in size from 64 bytes to 32 KB. To get a baseline value we also implemented a NOP program that simply received packets by multicast and then immediately re-multicast them. The results are shown in Figure 7 and are for a GSA running on a 125MHz, dual-processor SparcStation 20.

From the figure it can be seen that the GSA forwarding penalty (the difference between the two lines) is approximately 450μs. Almost all of this time is attributable to the cryptographic operations. Since we imagine that there will only be a few GSAs between any given sender and receiver, this penalty is unlikely to be significant for most applications. Note that because we use indirection in keying the penalty is fixed and does not increase with larger payloads.

10 Related Approaches

In this section, we review the (rather sparse) literature pertaining to multicast security protocols. These protocols are primarily multicast key distribution protocols. That is, they distribute a unchanging K_{GRP} to members as they join; they do not take care to change the key when the group membership changes.

Furthermore, most of these protocols are based on a centralized approach that basically works as follows. Each secure multicast group is assigned a group controller (GC). The GC processes all join requests and hands K_{GRP} to the joining member[12]. Protocols implementing this type of approach can be found in [25].

Also, Harney et al. describe the Group Key Management Protocol (GKMP) [15, 16]. GKMP is similar to the above, except that under GKMP the GC is implemented on a per-group basis, i.e., a designated member of the current group implements the GC.

This type of approach addresses neither the key management issues nor the scalability issues mentioned before. Indeed, the centralized GC only adds to the scalability problem.

More recently, Ballardie [3] has presented the scalable multicast key distribution scheme (SMKD) as part of the CBT multicasting architecture. In this proposal each multicast group has a group key distribution center (GKDC) similar to the GC that controls access to the group. It is suggested that, initially, the CBT core assumes the role of the GKDC. As nodes join the CBT distribution tree some of the functionality of the GKDC is "passed on" to the other nodes. In effect, each joining router is provided with a group access package that contains an ACL and K_{GRP}. When a node decides to join the multicast it contacts the nearest router acting as a GKDC which authenticates the joining member and passes it the group key.

Although SMKD does an admirable job of distributing K_{GRP} (by exploiting the implicit hierarchy of the multicast distribution tree), it too is unable to change K_{GRP} in the face of membership changes.

[12] Note the GC does not really need to be involved on leaves because the group keying material does not change on leaves.

Both GKMP and SMKD note that changing the keying material may be necessary when the group membership changes, but they offer no solution beyond establishing a new group. Of course, establishing a new group is clearly not scalable as it requires re-authentication on the part of all the group members.

11 Concluding Remarks

The primary motivation behind our design was enhanced scalability. In closing, however, it is interesting to note that the Iolus framework also provides certain other auxiliary benefits (both technical and non-technical in nature):

- **Enhanced Security.** The lack of a global K_{GRP} adds security in Iolus. While the compromise of a given K_{SGRP} is equally bad in that the attacker is given access to the group, it does not allow the attacker to arbitrarily share the K_{SGRP} with other principals. The other principals may be assigned to different GSAs making the compromised K_{SGRP} useless to them. In effect, subgrouping helps to contain attacks in some situations.

- **Flexible Management.** In a large multicast, it may easily be the case that no single entity (not even the GSC) will know each and every principal that should be allowed access to the secure multicast group. Subgrouping in Iolus allows a natural delegation of this responsibility.

  For example, consider a university that is offering a distance learning course which is multicast over the Internet. Some companies may wish to have their employees take this course. To do this, companies (using GSIs) apply to the university's GSC for admission to the secure multicast group (at the top level subgroup). Employees within a company then apply to their company's GSI for admission to the group through their company's subgroup. But, the GSC does not need to know exactly who is allowed in; it delegates that responsibility to the company's GSI which it trusts.

- **Flexible Pricing.** One of the primary reason for securing multicast communications is the economic value of the data being multicast. Flexible pricing is similar and related to flexible management. Consider, again, the university offering distance learning courses on the Internet. The university wishes to charge tuition, but instead of charging each principal individually, it charges the companies' GSIs at the top level subgroup. Having bought the multicast, the companies are then free to decide who should view it and how they should (or even if they should) pay for it.

In summary, multicast security is inherently different from unicast security. Instead of providing key management for a pair of principals, a multicast security protocol must provide key management for a changing group of principals. This creates a scalability problem because the keying material needs to be changed for the entire group whenever the group membership changes.

To address the scalability problems of secure multicast we have proposed Iolus: a scalable, general-purpose framework that can be used for either secure multicasting or multicast key management. Iolus discards the idea of large flat secure multicast group and replaces it with the notion of a secure distribution tree that is composed of multiple smaller secure multicast subgroups arranged in a hierarchy. Together these subgroups form a single virtual secure multicast group. The glue that holds the subgroups together consists of the group security agents that manage each subgroup. The GSAs "conspire" to invisibly deliver all multicast data securely to each of the subgroups, thereby creating a single secure multicast group image for the senders and receivers.

Furthermore, to better understand Iolus we have implemented a package of applications that use a protocol based on the Iolus framework. We have used the package to provide both a secure multicasting service and an independent key management service. Our preliminary experience suggests that Iolus-based protocols could be readily used with many applications and that the performance penalty imposed by the use of a secure distribution tree would not be significant for most applications.

Acknowledgements

I would like to thank Craig Partridge and Thomas Woo for many helpful discussions and for their extensive and insightful comments on various drafts of this paper. I am especially grateful to Craig Partridge for his early encouragement of this work. Finally, thanks are due to the anonymous referees for providing valuable feedback.

This work was supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship sponsored by the U.S. Air Force.

References

[1] T. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees: An Architecture for Scalable Inter-Domain Multicast Routing. In *Proceedings of the ACM SIGCOMM '93*, San Francisco, September 1993.

[2] T. Ballardie and J. Crowcroft. Multicast-specific security threats and counter-measures. In *Proceedings of the Symposium on Network and Distributed System Security*, San Diego, California, February 1995.

[3] T. Ballardie. *Scalable Multicast Key Distribution*. RFC 1949, May 1996.

[4] S. Berkovits. How to Broadcast a Secret. In *Advances in Cryptology: Proceedings of CRYPTO '91*, Lecture Notes in Computer Science **576**, Springer-Verlag, Berlin, 1991.

[5] M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. In *Advances in Cryptology: Proceedings of CRYPTO '94*, Lecture Notes in Computer Science **839**, Springer-Verlag, Berlin, 1994.

[6] G.H. Chiou and W.T. Chen. Secure Broadcasting Using the Secure Lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, August 1989.

[7] S.E. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of the ACM SIGCOMM '88*, Stanford, California, August 1988.

[8] S.E. Deering. *Host Extensions for IP Multicasting*. RFC 1112, August 1989.

[9] S.E. Deering. *Multicast Routing in a Datagram Internetworks*, Ph.D. Thesis, Stanford University, December 1991.

[10] S.E. Deering, D. Estrin, D. Farinacci, V. Jacobsen, L. Ching-Gung, and L. Wei, An Architecture for Wide-Area Multicasting. In *Proceedings of the ACM SIGCOMM '94*, London, September 1994.

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

[12] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. In *Proceedings of the ACM SIGCOMM '95*, Boston, August 1995.

[13] L. Gong and N. Shacham. Multicast Security and its extension to a mobile environment. *ACM-Baltzer Journal of Wireless Networks*, 1(3):281-295, October 1995.

[14] N. Haller and R. Atkinson. *On Internet Authentication*. RFC 1704, October 1994.

[15] H. Harney, C. Muckenhirn, and T. Rivers. *Group Key Management Protocol (GKMP) Architecture*. Internet Draft, September 1994.

[16] H. Harney, C. Muckenhirn, and T. Rivers. *Group Key Management Protocol (GKMP) Specification*. Internet Draft, September 1994.

[17] H.W. Holbrook, S.K. Singhal, and D.R. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *Proceedings of the ACM SIGCOMM '95*, Cambridge, Massachusetts, August 1995.

[18] I. Ingemarsson, D. Tang, and C. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5):714–720, September 1982.

[19] J.B. Lacy, D.P. Mitchell, and W.M. Schell. CryptoLib: Cryptography in Software. In *Proceedings of the USENIX UNIX Security Symposium IV*, Santa Clara, California, October 1993.

[20] S. McCanne and V. Jacobsen. vic: A Flexible Framework for Packet Video. In *Proceedings of the ACM Multimedia '95*, San Francisco, November 1995.

[21] National Bureau of Standards, U.S. Department of Commerce. *Data Encryption Standard*. FIPS Pub 46, Washington, D.C., January 1977.

[22] C. Partridge, T. Mendez, and W. Milliken. *Host Anycasting Service*. RFC 1546, November 1993.

[23] R.L. Rivest. *The MD5 Message-Digest Algorithm*. RFC 1321, April 1992.

[24] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, New Delhi, March 1996.

[25] L.C.N. Tseung. Guaranteed, Reliable, Secure Broadcast Networks. *IEEE Network Magazine*, 6(3), November 1989.