A Survey of Security Issues in Multicast Communications

Matthew J. Moyer, Georgia Institute of Technology
Josyula R. Rao and Pankaj Rohatgi, IBM Thomas J. Watson Research Center

Abstract

The recent emergence and popularity of group-oriented applications on the World Wide Web has triggered a demand for scalable security solutions for group communication. One such solution, secure multicast, is appealing because it leverages the efficiency of multicast data delivery. However, it also presents several research challenges, most notably in group communication architecture, group key management, and message source authentication. In this survey, we discuss these issues and review proposed solutions to them.

The recent growth of the World Wide Web has sparked new research into using the Internet for novel types of group communication, like multiparty videoconferencing and real-time push-based information delivery systems such as stock quote services. These applications require multicast to minimize the volume of network traffic they generate. Conceptually, multicast-enabled routers take each packet sent over a multicast channel and route it to every receiver listening on that channel. The set of principals sending and/or receiving data on a particular multicast channel is called a multicast group. Multicast’s major advantage over unicast is that it allows the sender to send each packet just once; the routers automatically forward the packet to each receiver that wants it, while minimizing the number of copies of the packet that traverse the network.

The traditional mechanism used to support multicast communication is IP multicast. In IPv4, the class D addresses (ranging from 224.0.0.0 through 239.255.255.255) are reserved for multicast communication. In addition, multicast-enabled hosts and routers participate in the Internet Group Management Protocol (IGMP) [1] to maintain a membership information, and multicast-enabled routers also participate in one or more multicast routing protocols [2]. Currently, IP multicast is only accessible via UDP. Any host can send a UDP packet to a multicast address, and the underlying multicast routing mechanism will deliver the packet to all recipients who have explicitly joined that multicast group and are within the time-to-live (TTL) scope of that packet.

Multicast has the potential to be very useful, but it suffers from many problems stemming from the inherent complexity of routing packets to a large group of receivers. In particular, guaranteeing reliability (the timely, in-order delivery of each packet to every receiver in a group) and security (the notion that only the registered members of a multicast group can send packets to the group or receive packets sent to the group) is very difficult. Both of these issues encompass many challenging research problems. In this article we explain the major research challenges of multicast security and highlight important contributions that have been made in this field. The next section describes the difficult research issues secure multicast presents. We then describe the issues in proposing an architecture for secure multicast and designing a key management solution. Then, in three sections we describe some of the best current solutions to each major problem. We discuss some related work, and the last section concludes the article.

Security Issues in Multicast

The objectives of a multicast security infrastructure are simple: preserve authentication and secrecy for all group communication so that only registered senders can send packets to the group and only registered receivers can read packets sent to the group. Due to the lack of network-level access control in the Internet, enforcing message secrecy for a multicast group requires data encryption. This requires a group key management solution to distribute and maintain cryptographic keys with registered group members. Similarly, cryptographic authentication schemes are necessary to ensure that registered receivers can verify that received packets come from registered senders. Most research on secure group communication has focused on the architecture of secure groups and the problem of group key management; however, recent research has also focused on efficient packet authentication. This article surveys all three of these areas.

Issues in Architecting Key Management Solutions

We begin discussion of the issues in key management by introducing a simple solution to group key management,
examining its weaknesses, and listing a set of criteria by which to compare other solutions with it.

Inadequacy of Naive Group Key Management

The naive solution to the group key management problem is straightforward. We establish a centralized group controller that manages keys for the group. The controller communicates the group keys to each group member individually via a secure unicast connection. Distributing a key to a group requires communication costs linear with the number of group members. The simplicity of this solution is attractive; however, its inefficiency renders it inadequate for most multicast applications, for two reasons.

First, we must distribute new group keys every time a party joins or leaves the group. To protect the secrecy and integrity of past messages, the group must change its keys every time a new member joins. Also, to protect the secrecy and integrity of future messages, the group must change its keys every time a current member leaves. The only parties who may know the current keys are those currently participating in the group communication. For large multicast groups with highly dynamic membership, it is infeasible to perform a linear-cost key update every time a member joins or leaves; therefore, we require a key management solution that incurs a sublinear messaging cost on key updates.

Second, various systems and networking issues make it difficult to consistently route packets from a single source to a large, widely dispersed group of receivers in a timely manner. The first potential hazard is failure of the group controller node, which is fatal to the group in any centralized solution. The second problem that can occur is a network failure. If there is a high traffic load or a partition somewhere in the network, some members may receive key updates more quickly than others, or may miss a key update altogether. This not only inhibits secure communication among legitimate members of the group, but also creates potential security vulnerabilities in which past group members can exploit the fact that not all current members are using the same key. A malicious party could send illegitimate messages encrypted using stale keys to the group, or could coax some current members into sending legitimate messages to the group under stale keys. Clearly, a centralized key management solution is not a robust design. Systems and networks often slow down or fail; we need an architecture that can deal effectively with these realities.

Evaluation Criteria for Key Management Solutions

Based on the above observations, we present and evaluate alternative solutions to group key management for secure multicast groups. Before we discuss each solution, however, we present a list of criteria useful for examining and comparing various solutions. Each criterion is listed and briefly explained:

- System architecture: An ideal system might degrade gracefully in the face of network and system failures, but should not fail entirely. Various hierarchical and distributed architectures can provide this level of availability; centralized solutions (in particular, the naive solution) cannot.
- Recovery from failure: A well-designed system should not only withstand various node and network failures, but also recover from most system failures without having to reinitialize the entire group.
- Scalability: The solution should scale to handle very large, widely distributed groups. It should also be capable of handling very frequent key updates, to accommodate groups with highly dynamic membership.
- Number of keys with a controller: The naive solution requires the group controller to store $n + 1$ keys: one pairwise key with each group member, plus a shared group key. Other solutions exist in which controllers store more keys to accommodate more efficient key management algorithms.
- Number of keys with each group member: The naive solution requires members to store a constant number of keys, but its bandwidth requirements are unacceptable. We later describe alternative solutions that store more keys at each member but achieve superior bandwidth efficiency.
- Join and leave secrecy: In group communication, join secrecy ensures that new group members cannot read past messages, and leave secrecy ensures that past group members cannot read current or future messages. In general, join secrecy is easy to achieve efficiently; the group key can be updated upon each new join by multicasting a fresh key to the group encrypted with the old group key. The naive solution preserves both join secrecy and leave secrecy; however, preserving leave secrecy using the naive solution requires a linear-cost key update every time a member leaves the group. Some alternative solutions improve key update efficiency by not always updating keys on a leave. There are some applications in which this behavior is acceptable, but in general it is desirable to guarantee both join and leave secrecy.
- Number of messages to update keys on a join: The naive solution updates the group key on a join using $n$ messages: one for each group member. Some alternative solutions update the group key on a join in constant message bandwidth by using multicast, while others trade slightly more expensive key distribution on a join for improved key distribution on a leave.
- Number of messages to update keys on a leave: When a member leaves the group, the naive solution updates the group key using $n$ messages: one for each remaining group member. We describe alternative solutions that use combinations of multicast messages, physical and virtual subgroups of members, and hierarchies of auxiliary keys to reduce this number to sublinear in the size of the group.
- Processing time for key management: In the naive solution, when a member joins or leaves the group, the group controller must generate a new key, and then encrypt and sign $n$ messages containing that key. Alternative solutions achieve better cryptographic processing performance, at the cost of a more complex key management structure.
- Protection against collusion: The naive solution ensures that nonmembers of the group cannot collude and break the group's encryption scheme. There are, however, other key management solutions that sacrifice strength against collusion in favor of more efficient key management. In general, it is desirable to ensure that collusion among an arbitrary number of nonmembers cannot occur.
- Reliability requirements: The naive solution uses unicast to distribute keys to group members. It is therefore easy to detect and correct the situation in which a message misses a key update message. Other schemes use multicast to distribute key update messages more efficiently, but IP multicast is unreliable (best-effort), so systems that use multicast to distribute keys must employ a reliable multicast subsystem.
- Dependence on routing protocols: At this point, not all Internet routers support IP multicast, and there exist many different multicast routing protocols. It is unclear which multicast protocol(s) will emerge over the long run as the Internet standard. Therefore, it is most desirable for a multicast security infrastructure to remain completely independent of underlying multicast routing protocols. There exist solutions, however, that depend on a particular multicast

---

1 For additional information about multicast security issues and evaluation criteria, we refer the reader to [3-5].
infrastructure. These solutions often achieve better performance than more modular systems, at the cost of flexibility. Applicati...s for which general-purpose solutions are unnecessary and inefficient.

The Outline and Scope of This Article
In the remainder of this article, we summarize important research in secure multicast, including work in progress and prototype systems. We focus on secure multicast architectures, group key management, and packet source authentication. Our summary consists of a high-level overview of each contribution. Interested readers are encouraged to consult the original sources to learn more.

Secure Multicast Architectures
The inherent complexity of routing packets securely to a large, widely distributed group of receivers has motivated the design of several architectural frameworks for secure multicast. In this section we introduce and evaluate three such frameworks.

Iolus
Iolus [5], a high-level infrastructure for secure multicast, is useful as either a standalone group key management service for secure multicast or a security module in a multicast application. We first describe the major features of Iolus; then we analyze its strengths and weaknesses.

Architectural Framework — Iolus addresses the problem of efficient key updates and reliable data transmission by partitioning a multicast group into a hierarchy of subgroups, each with relatively few members and its own multicast address. The architecture uses a secure distribution tree to create the illusion of a single group to the members of each subgroup. The tree is composed of group security agents (GSAs), trusted entities that coordinate packet routing and manage security for the group. The GSA at the root of the tree is called the group security controller (GSC), and the other GSAs are called group security intermediaries (GSI’s). Each subgroup has its own cryptographic keys. The GSI in charge of a subgroup serves two purposes:

- Manage its subgroup’s keys
- Mediate all communication between its subgroup and other subgroups

GSIs and subgroups can exist on multiple levels, and lower-level GSIs act as clients for higher-level GSIs. Figure 1 shows an example of a secure distribution tree.

Operational Semantics — This section provides a high-level overview of the most important operational details of Iolus, including the algorithms used to initialize the group, add and delete members, and transmit messages.

- Group initialization and member addition: The initialization of a multicast group under Iolus requires only that the GSC and GSIs be running, and the GSIs know the address of the GSC. At first, none of the GSIs are actually in the multicast group. When an authorized party tells the GSI it wants to join, the GSI generates a new subgroup key and sends it to the new member via secure unicast. Then the GSI joins the next highest subgroup in the hierarchy. The process by which a GSI joins a higher-level group is conceptually the same as the process used when a client joins a bottom-level subgroup. This process continues up the tree until the appropriate top-level GSI has contacted the GSC and joined the multicast group. If a GSI is already a member of the multicast group when it receives a join request from a member or a lower-level GSI, it does not need to continue the recursive join process. Also, if there are already members in the subgroup that the GSI manages, it must change the subgroup key and distribute it to all members on every join. The GSI distributes the new subgroup key to the existing group members by encrypting it under the old subgroup key and multicasting it. Thus, adding a new member to an existing group requires only two messages: one unicast message to the new member and one multicast message to the existing members.
- Member deletion: Deletion of a group member can occur if a member explicitly asks the GSI to be removed, or the GSI removes him forcibly. If the member being removed is the only member of the subgroup, the GSI contacts its parent GSI and removes itself from the secure distribution tree. But if there are members in the subgroup other than the one being removed, the GSI must instead generate a new subgroup key and distribute that key to all remaining members of its subgroup. This procedure incurs bandwidth costs that are linear with the size of the subgroup, but less than the size of the entire multicast group.
- Data transmission: Since each subgroup uses a different key, the GSIs are responsible for translating data from one key to another and routing it to other GSIs as appropriate. For example, in Fig. 2, if a client in subgroup 1 sends a message to the group, the GSC encrypts it under the encryption key for subgroup 1. All members of subgroup 1 know this key, including GSI; however, nobody outside subgroup 1 knows it. To forward the message to other subgroups, GSI; decrypts it, re-encrypts it with the encryption key for subgroup 12, and sends it to the members of subgroup 12. Upon receiving the message, GSI; decrypts it, re-encrypts it with the encryption key for subgroup 2, and sends it to the clients in subgroup 2.

Strengths and Weaknesses — The major benefits of using a secure distribution tree are twofold. First, this architecture localizes the effect of group membership changes to one subgroup. Even when using the naive key management scheme, changing subgroup keys is not very expensive, because each

---

2 In practice, GSIs do not encrypt and decrypt entire messages during the translation process. Instead, they encrypt each message under a random key K, and then encrypt K under a key encrypting key (KEK) and attach it to the message. The key translation process consists only of translating K from one KEK to another.
subgroup is small. The second benefit of a secure distribution tree is that it distributes the security over many nodes, eliminating any single point of failure except the top-level GSC. If a GSI experiences a system failure or a security breach, only its subgroup loses service; the rest of the group can keep functioning.

The Iulus framework has many positive features; however, it also suffers from several drawbacks. First, Iulus requires substantial resource overhead to manage a multicast group. A very large group could have many GSIs, and even if enough machines were available on which to deploy all of the GSIs, the performance of the system might still be less than desirable due to the inherent complexity of the Iulus hierarchy. Second, even though Iulus uses a hierarchical system to distribute the group management workload over many nodes, it still relies on a single top-level node, the GSC, for correct operation. If the GSC fails, many of the subgroups will be cut off from each other. Despite these problems, however, the Iulus architecture contributes several useful techniques to secure multicast research.

The Nortel Framework

The second architecture we address comes from a pair of Internet drafts published by Nortel. Reference [6] specifies a high-level architectural framework without discussing the details of key management, and [7] describes one particular solution for key management that can be used within the architecture. We summarize both drafts here.

High-level Architectural Framework — The Nortel architecture consists of a two-level hierarchy of regions. At the highest level is the trunk region; it is bordered by many leaf regions. Each leaf region has a border key manager which is also a member of the trunk region. Figure 3 illustrates this design. The border key managers for each leaf region participate in a key management protocol for a trunk key. The trunk key allows the border key manager for each leaf region to act as a message translator, sending and receiving messages to and from other leaf regions. The details of trunk key management are unspecified; however, in practice, the trunk key could be managed by a centralized trunk controller or negotiated between the border key managers. A robust key negotiation algorithm for the trunk key can make this architecture very robust against failures at any of the border key managers. We do not discuss the trunk level in any more detail. In the next section we examine the key management infrastructure within each leaf region.

Intraleaf Key Management — We noted previously that partitioning a multicast group into subgroups (leaf regions) local-
izes the effect of membership changes. Another benefit of this architecture is that it allows each leaf region to employ its own key management techniques. In practice, any key management scheme can be used within each leaf; however, Nortel has proposed a specific key management infrastructure for leaf regions. We describe it here.

Inside each leaf reside the following logical components: a set of key managers, one or more multicast-enabled routers, a key translator, and some number of multicast group members. The highest-level key manager in a leaf region is the domain key distributor (DKD). The DKD manages the key for a set of area key distributors (AKDs); each AKD acts as a key manager for a subset of the multicast group members that reside in the leaf region. The purpose of the DKD and AKDs is to further localize the effect of group membership changes. Figure 4 shows the logical structure of a leaf.

An interesting feature of Nortel's leaf-level key management scheme is that it consists of two planes: a control plane and a data plane. Within a leaf, the DKD and AKDs reside in the control plane. The DKD manages a key called the ALL-KD-key; it is shared among the DKD and all AKDs in that leaf. Each AKD manages an Area-Group-Key that is shared among itself and all members in its subgroup. But none of these keys is used to actually encrypt application data. The application data for the entire leaf is encrypted under a special multicast key (MKey) that is part of the data plane. The border key manager translates application data to and from the trunk between the trunk key and the leaf MKey. The purpose of the control plane is to efficiently distribute the MKey for a leaf to all members of the multicast group in that leaf. Thus, anytime a member in the leaf joins or leaves the multicast group, all group members in that leaf must receive a new MKey. This may seem expensive, but since the membership change is confined to just one AKD's area, the AKDs in all other areas can multicast the new MKey to members in their area, encrypted under their area group key. The only area in which the key management semantics are complicated is that in which the membership change occurred. Assuming that the number of members in each area remains relatively small, this is not very expensive, even under a naive key management scheme.

![Figure 4. Nortel's intradomain key management framework.](image)

**Strengths and Weaknesses** — The Nortel framework is similar to Iolus in that its hierarchical structure minimizes the cost of membership changes and makes it more robust against node failures. It differs from Iolus, however, in that its top-level (trunk) region may be fully distributed, thus potentially providing more robustness than Iolus against high-level node failures. In most other aspects, this framework has the same strengths and weaknesses as Iolus.

**The SRM Toolkit**

Chung et al. [8] have built a Java prototype of a toolkit for secure reliable multicast (SRM) that derives ideas from both Iolus and the Nortel framework. This section summarizes the important details of their system.

**Overall Structure** — The SRM architecture consists of the following components:

- A **directory server**
- Several **domain controllers**, analogous to the GSIs in Iolus
- A **reflector** for each domain that handles interdomain packet routing
- **Clients** that use the services provided by the other system components

This system uses a two-level hierarchy, similar to the Nortel framework, to partition the multicast group into domains. A domain controller manages the shared key for the clients within a domain. Each domain also has a reflector collocated with the domain controller. It translates and routes packets to and from the other domains in the group, similar to the way GSIs translate and route packets in Iolus. At a higher level, there is a top-level configuration server, also called the **master controller**, that manages the domain controllers. The directory server implements a group directory service; it uses Lightweight Directory Access Protocol (LDAP) [9] to answer queries from clients about which domain controller to contact when joining a multicast group. Thus, it must store a database of domain controllers for the group, as well as an access control list (ACL) of clients who may join the group. Domain controllers for a group contact the directory server at startup to download the ACL for their group and the list of addresses of the other domain controllers for the group.

**Client Registration and De-registration** — To join a group, a client first contacts the directory server and obtains the address of a domain controller for the group. Then the client contacts the domain controller over a secure unicast channel and authenticates himself. If the authentication succeeds, the domain controller updates the shared keys for the domain and sends the appropriate messages to the new member and current members of the domain. Removing a client is similar to adding a client. The removal occurs by updating the shared keys for the domain and multicasting the appropriate messages to the clients in the domain. We defer the details of the SRM key management structure until later.

**Strengths and Weaknesses** — The SRM infrastructure has many of the same strengths and weaknesses as Nortel’s architecture. Its two-level framework is simpler, but potentially less
scalable, than Iolus' multilevel structure. Also, unlike the Nortel framework, it suffers from a single point of failure at the master controller, as does Iolus.

Comparison of Techniques
Table 1 summarizes the important characteristics of the three architectures we have discussed.

Group Key Management Techniques
The previous section focused on group architectures, but it also described some of the key management schemes that have been proposed for those architectures. In this section we turn our attention specifically to algorithms and structures that permit efficient multicast group key management. Some of the schemes we discuss are not appropriate in all circumstances, and most require reliable multicast. Nevertheless, they are interesting and may prove useful, especially for subgroup key management. We begin this discussion by reviewing the naive key management scheme; then we introduce more efficient strategies based on logical key hierarchies and one-way function trees. Finally, we mention one scheme that achieves even better efficiency at the expense of collision sensitivity and join/leave secrecy.

The Naive Key Management Scheme Revisited
A centralized group controller shares a secret key with each of \( n \) members. On each join or leave, the controller changes the group key by securely uncasting an update to each member. Each client stores two keys (a shared key with the controller and the group key), and the controller stores \( n + 1 \) keys (one key for each client, plus the group key). This is the naive key management scheme.

There are many noteworthy positive features of this scheme. First, it is simple and easy to implement. Second, its key storage requirements are reasonable if we assume that the group controller runs on a powerful machine. Third, this scheme guarantees join and leave secrecy and is resistant to arbitrary collusion among nonmembers of the group. Finally, this scheme does not require any specialized, underlying infrastructure such as reliable multicast. In spite of these advantages, however, this model scales poorly in terms of both group size and group dynamics. The network load and the group controller's processing load on a join or leave are both linearly proportional to the group size. Thus, despite its many desirable properties, the naive key management scheme is impractical for all but the smallest groups.

Tree-Based Key Management
We now introduce an alternative key management technique that uses a more complex key management infrastructure and some additional assumptions in order to reduce the cost of key updates significantly. Both Wallner et al. [10] and Wong et al. [11] have observed that by using a logical hierarchy of key-encrypting keys and reliable multicast, we can manage a group key such that all key updates incur messaging and computational cost logarithmic to the size of the group. Wong et al. have performed extensive theoretical and experimental analysis on various types of key graphs; they have concluded that the most efficient key graph for group key management is a \( k \)-ary tree. Wallner et al. have described a key management structure that uses a binary tree in the same fashion. In this section we describe and analyze the more general case of a \( k \)-ary tree; the binary tree scheme is a straightforward specialization of the more general case.

Tree-based key management works as follows. A multicast group has \( n \) members, \( M_1 \) through \( M_n \), and a centralized group controller. A member joins the group by contacting the controller via a secure unicast channel. At the time the new member joins, he and the controller negotiate a pairwise secret key. The controller stores a \( k \)-ary tree structure in which each node contains a key. At the leaves of the tree are the \( n \) secret keys the controller has negotiated with each of the members in the group. Each member stores a subset of the controller's keys. The subset of keys stored by member \( j \) is the same as the set of keys in the path from leaf \( j \) to the root of the tree, including leaf \( j \) and the root itself. The root node stores the group's shared key; all other keys in the tree are auxiliary keys, used only to facilitate efficient key updates. The total number of keys stored by the controller is approximately \( kn - 1k - 1 \), and the total number of keys stored by each member is \( \log_{k} n \). Figure 5 illustrates this tree for \( k = 3 \) and \( n = 9 \).

We now describe how keys are updated when a member joins or leaves the group. In Fig. 5, assume that member \( M_8 \) leaves the group. When the group's membership changes, all keys known by the member who joined or left the group must be updated, except for the secret key shared between the controller and the joining or leaving member. In this example, we must update keys \( K_{7} \) and \( K_{11} \). We can use the existing key hierarchy, along with reliable multicast, to efficiently dis-

![Figure 5. A k-ary key management tree for k = 3, n = 9.](image-url)
of applying a one-way function $g$ to $k_x$, that is, $k'_x = g(k_x)$. (It is computationally infeasible to find an inverse function $g^{-1}$ that would map any $k'_x$ back to the corresponding $k_x$.) $g$ can be well known to both group members and nonmembers. Each group member knows the unblinded key for every node on the path from its leaf to the root. In addition, each member knows the blinded key for every node that is a sibling of a node on the path from its leaf to the root. Figure 6 shows an example OFT tree, highlighting the blinded and unblinded keys that are known to a particular group member.

The purpose of blinded and unblinded keys is to allow group members to compute higher-level keys from lower-level keys, thus reducing the number of keys the group controller must explicitly send during key updates. Each interior (nonleaf) node in the tree contains a derived key, and keys are derived as follows. Suppose that node $x$ is an interior node, and that its left and right children are $x_L$ and $x_R$, respectively. The unblinded key for $x$, denoted $k_x$, is the result of applying some mixing function $f$ to the blinded keys for $x_L$ and $x_R$. Symbolically, $k_x = f(k_{x_L}'$, $k_{x_R}')$. $f$ can be a simple XOR function; it does not have to be one-way like $g$, and it can be well known. For every node $x$, the group members that possess $k_x$ can use $g$ to compute $k'_x$ from $k_x$. These members also know $k_{x_L}'$, the blinded key of node $x_L$, which is the sibling of node $x$. Then, using $f$, they can combine $k'_x$ and $k_{x_L}'$ to compute $k_x$, the unblinded key of node $x$, which is the parent of nodes $x_L$ and $x_R$. By this process, every member can compute all necessary keys, including the root key.

When a member joins or leaves the group, the controller must change all keys along the path from the leaf where the change occurred to the root. But since all keys at interior nodes are derived from other keys, the only new key that the controller must explicitly generate is the affected leaf. Then the controller must reconstruct the derived keys in the tree based on the new leaf key. After deriving the new tree of keys, the controller must securely multicast enough key information to the group so that all current group members can recompute any keys that have changed as a result of the membership change. This process requires the controller to multicast $\log_2 n$ key updates, where $n$ is the size of the group. The actual messages sent are the blinded keys for the siblings of the nodes along the path from the affected leaf node to the root of the tree. These keys are encrypted under existing keys such that only those members who should receive the updates have the keys necessary to decrypt each message.

The OFT key management scheme achieves performance similar to that of the tree-based scheme in the previous section, with a few subtle differences. First, it reduces the number of messages required on a key update to $\log_2 n$. Second, it requires each group member to perform some local computation to derive higher-level keys from lower-level blinded and unblinded keys, whereas the standard tree-based scheme does not require any computation by the group members. Third, depending on which mixing function $f$ is used to compute higher-level unblinded keys from lower-level blinded keys, this scheme may require group members to store $2\log_2 n$ keys; all of the unblinded keys along the member's path to the root, and blinded keys of the siblings of those nodes. (In comparison, the standard tree-based scheme requires each member to store only $\log_2 n$ keys.)

---

3 These performance figures assume that the tree remains balanced. If the tree is not balanced, these costs are not logarithmic, but linear with the size of the group. Meyer et al. [12] have investigated and implemented techniques for maintaining and balancing a binary key management tree.

4 Just as in the standard tree-based key management scheme, this performance figure assumes that the OFT tree is balanced.
the blinded keys at the siblings may not be necessary, however, if \( f \) is chosen such that \( y \) can be computed from \( x \) and \( z \), where \( x = f(y, z) \). (The XOR function has this property.) Otherwise, the ORF key management scheme is conceptually no different from the standard tree-based scheme; thus, its basic strengths and weaknesses are similar to those discussed above.

**SRM Key Management**

The SRM toolkit described earlier uses a novel approach to key management. Its scheme is not practical, because it can be broken under certain circumstances, as we describe. It is interesting, however, for the insight it provides into the group key management problem. We explain the scheme here.

Let \( n \) be the number of members in a domain. The domain controller stores \( 2 \log n + 1 \) keys, and each member knows a unique subset of \( \log n \) + 1 of these keys. The keys are logically arranged in two lists, each of size \( \log n \). The remaining key is the group key, which all members know. Figure 7 shows an example of the keys stored by the controller under this scheme for \( n = 32 \). To delete member \( j \) from the domain, the domain controller updates the group key and multicasts it encrypted under each of the keys that \( j \) does not know. In Fig. 7, suppose that member \( j \) knows keys 1a, 2a, 3a, 4a, and 5a; the domain controller would encrypt the new group key under keys 1b, 2a, 3a, 4b, and 5a, so that every member except member \( j \) can decrypt at least one message containing the new group key. This key management method stores fewer keys than the tree-based scheme discussed above, and also removes a member from the group with fewer messages; however, it is insecure because it becomes vulnerable to collusion and fails to guarantee join or leave secrecy after the first member is deleted.

To understand this scheme’s flaw, consider the case in which two members are deleted from the domain in succession. Of the \( 2 \log 2 \) keys in the domain controller’s two lists, both members know exactly half, but they do not know the same set of keys. Thus, each member knows at least one key the other does not know. When deleting the first member from the group, the domain controller multicasts the group key encrypted with all of the keys the deleted member does not know. Then, to delete the second member, the domain controller again updates the group key and multicasts it encrypted with all of the keys the second member does not know. But the first deleted member must know one of the keys the second member does not know; thus, the first deleted member can decrypt the new group key and eavesdrop on the future communication after one additional member is deleted.

This key management scheme is interesting because it tries to use fewer than \( n \) keys to manage a group of size \( n \). It is provable that any scheme that does this will be vulnerable to collusion. To see this, note that any such group must have a proper subset of members \( S \) who know all keys; in particular, the users in \( S \) know all the keys known by some other group member \( G \) who is not in \( S \). Clearly, the key management scheme cannot remove all members of \( S \) and yet retain \( G \) in the face of collusion between members of \( S \).

**Comparison of Techniques**

Table 2 outlines the basic facts about each key management scheme discussed in this section.

---

![Figure 7. The SRM key management system.](image)

**Packet Source Authentication**

We now discuss the third active area of secure multicast research. **Packet source authentication** (i.e., authenticating the sender of a received packet) is a fundamental security issue in multicast. There are three levels of source authentication; we describe them here in increasing order of complexity. In the first and most basic level, a registered receiver can verify that a packet was sent by a registered group member (either a registered sender or a registered receiver) without necessarily knowing the identity of the sender. This level of authentication allows a receiver to know that the packet has not been injected by someone outside the secure group. There is thus a valid and efficient cryptographic solution for achieving this level of authentication: the use of a shared secret-key-based message authentication code (MAC). This same technique is widely used in unicast authentication (e.g., in IPSec) [14, 15]. The shared secret MAC key could be distributed using the same group key management scheme as for the data encryption keys. However, this level of authentication, and particularly its implementation using MACs, is inadequate in most multicast settings, because MACs can be generated by anyone with access to the secret key. In particular, the MAC-based solution allows any registered receiver to masquerade as a registered sender. This could result in a significant security exposure in large group settings. In the second level of authentication, a registered receiver can verify that a packet was sent by a registered sender. This allows the receiver to ascertain that the packet was sent by some registered sender and not by a registered receiver or some other entity. The third and most desirable form of authentication allows registered receivers to precisely identify the registered sender who sent each received packet. Achieving these

---

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Naive</th>
<th>Tree-based</th>
<th>OFT</th>
<th>SRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized architecture?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of controller keys</td>
<td>( n + c )</td>
<td>( (n + c) / 2 )</td>
<td>( 2h + n )</td>
<td>( 2 \log n + 1 )</td>
</tr>
<tr>
<td>Number of member keys</td>
<td>( c )</td>
<td>( \log n )</td>
<td>( 2 \log n )</td>
<td>( \log n + 1 )</td>
</tr>
<tr>
<td>Mgs sent on join/leave</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>Computation at controller on join/leave</td>
<td>( O(n) )</td>
<td>( O(\log n) )</td>
<td>( O(\log n) )</td>
<td>( O(\log n) )</td>
</tr>
<tr>
<td>Computation at member on join/leave</td>
<td>( O(1) )</td>
<td>( O(\log n) )</td>
<td>( O(\log n) )</td>
<td>( O(\log n) )</td>
</tr>
<tr>
<td>Vulnerable to collusion?</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Reliable multicasts?</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Join secrecy?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Leave secrecy?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

* In the naive case these are unicast messages; in all other cases these are multicast messages.

**Table 2: A comparison of secure multicast key management schemes.**
more desirable forms of source authentication efficiently is an active area of research. In the remainder of this section, we present criteria for judging source authentication solutions. We then discuss several solutions and evaluate them based on the criteria.

Criteria for Judging Source Authentication Solutions

There are several criteria by which one can judge source authentication schemes. Briefly, they are:

- **Efficiency**: Since every packet must be authenticated, the authentication information should be inexpensive to generate and verify. In addition, since the authentication information must be placed in each packet, the size of this information should be small.
- **Reliability requirements**: Certain authentication schemes require some form of reliability from the underlying transport mechanism. The reliability requirement can vary from absolute reliability (i.e., in-sequence ordering of packets with no packet loss) to partial reliability, which requires that most packets be received. IP multicast is a best-effort service, and current solutions for reliable multicast are complex, expensive to deploy, and not yet standardized. Thus, reliability is an important evaluation criterion.
- **Vulnerability to collusion**: Ideally, an authentication scheme should resist arbitrary collusion among group members. However, there are classes of efficient authentication schemes that are parameterized on the number of colluders they resist; these may be useful in certain scenarios.
- **Latency**: Some schemes require the sender and/or receiver to collect several data packets together before authenticating them. This may not be an issue in smooth unidirectional flows, but is an important issue for highly interactive applications in which packets must be sent as soon as they are generated.

The Naive Authentication Solution

The most intuitive solution to the authentication problem uses public key signatures on each data packet. To achieve group sender authentication (i.e., assurance that each packet comes from a registered sender in the group), all senders can share a private key in a digital signature scheme. To achieve individual sender authentication, each sender could have a separate private signature key. This solution is desirable because it works in the fully unreliable setting, is immune to collusion attacks, and does not introduce latency. However, there are major problems with the performance of this approach. Public key signatures using acceptable algorithms and key lengths are very expensive, in terms of key generation and signature verification overhead. For example, even a high-end workstation can generate only about 50 1024-bit RSA signatures/s. Some multicast applications require packet rates exceeding 50 packets/s, and most applications require less throughput cannot afford to devote a large fraction of CPU cycles to packet signing. Also, the size of a private key signature is generally much larger than that of a MAC; this could be a problem in some settings. Thus, the public key signature-per-packet solution is not practical. This realization has spurred research into more efficient authentication schemes. We discuss some of these schemes below.

The Multiple MACs Scheme

One alternative to the naive solution is to simply relax the security requirements for authentication. For example, one could use more efficient but less widely studied signature algorithms. This approach could provide fast, secure authentication if the underlying cryptographic assumptions are valid. However, if these assumptions turn out to be invalid, any scheme that relies on them could be easy to break. Therefore, this type of solution suffers from the major drawback that it is hard to quantify the amount of effort required to break the scheme.

An efficient approach to multicast packet authentication which relaxes the security requirements but does not suffer from this drawback was proposed in [14]; this concept is called an asymmetric MAC. The basic idea in this approach is that the sender knows several secret keys, and these keys are shared with the recipients to maintain several properties of the subsets of keys held by the recipients. One such property could be that no subgroup of \( N \) receivers should know all the keys known by any other receiver. The server can authenticate a message by computing MACs using all its secret keys and appending all these MACs to the message. This collection of MACs is known as an asymmetric MAC. Each recipient can verify the parts of the asymmetric MAC for which it knows the secret keys; if all these MACs are valid, the receiver accepts the message as genuine. This solution is computationally efficient, works in a fully unreliable setting, and introduces no latency. Note that a receiver cannot by itself forge an asymmetric MAC, because it does not know all the keys of a sender or even all the keys known to some other recipient. Based on the subgroup property described above, even \( N \) recipients cannot forge an asymmetric MAC to fool some other recipient. But if there are more than \( N \) colluders, the security of the scheme could break down. Furthermore, if there are enough colluders to know all the sender keys, the scheme breaks down completely.

It is easy to see that the number of MACs computed by the sender has to be a linear function of the number of colluders for the scheme to be resilient. The advantages of this approach are that it can employ well-studied and cryptographically secure MAC schemes, and it can remain secure until the limit on the number of colluders has been reached; therefore, its security is easy to quantify. But even though this scheme is useful in many scenarios (e.g., when groups are small and collusion can be controlled), it does not work well when the multicast group is very large, or large collisions are likely to occur and difficult to detect. Another problem with this scheme is that the number of MACs that must be appended to guard against a large group of colluders can be large, which could introduce substantial bandwidth overhead on each packet. To address this issue, [14] also introduces another variant of the scheme which uses more MACs, each only 1 bit long. This drastically reduces the size requirements, with only a minor effect on the security of the scheme. (Forgers are now possible even without \( N \) colluders, but only with exceedingly small probability.)

The Stream Signing Solution

If reliability of transmission is not an issue, there is another approach known as stream signing [16] that could be used to sign the multicast packets efficiently and provide the security guarantees associated with digital signatures. In this approach only one regular signature is transmitted at the beginning of a stream, and each packet contains either a cryptographic hash of the next packet in the stream or a one-time public key that can verify the one-time private key signature on the next packet. This approach, however, cannot tolerate a lost packet. While this may not be an issue in reliable Internet protocols (e.g., those based on TCP/IP), it is a major issue for multicast applications that use UDP over IP multicast. Another problem with stream signing is that if the stream being sent is not known to the sender in advance, the sender needs to embed one-time keys and signatures into the packet stream. These keys and signatures are fairly large and can result in substantial space overhead within each packet. Some of the overhead can be reduced if the sender is allowed to introduce latency, but this is not a viable option for peer-to-peer interactive multicast applications such as distributed simulations and gaming.
The Wong-Lam Scheme

In [17] a different approach is proposed for packet authentication when a sender is allowed to introduce latency and group together several consecutive packets. Essentially, this approach forms an authentication tree [18] from packets collected during a time interval and signs the root of the authentication tree. Each packet is augmented with the signature on the root and ancillary information which includes hashes on the logarithmically many nodes on the authentication tree. This allows each packet to be individually verified, so the scheme works in the fully unreliable setting. The scheme is essentially a signature scheme and is therefore immune to collusion attacks.

This approach is quite effective in client-server scenarios where latency is not an issue and the server is dedicated to serving a small number of multicast flows, each with reasonably smooth flow rates and strictly enforced bounds on processor loading. For this reason, it has also been proposed that this scheme be used to provide authentication for Real-Time Transport Protocol (RTP) for audio and videoconferencing [19]. However, for more general use this approach suffers from several practical drawbacks. First, as discussed earlier, delaying and grouping of sender packets is not possible for peer-to-peer interactive application such as distributed simulations and games. Second, there is the problem of serving multiple multicast flows. This is best illustrated by the following example. Suppose a server only has enough cycles to perform 10 public key operations/s. Using the scheme in [17], such a server could potentially serve a single smooth flow of hundreds of authenticated packets per second with only a minor delay of a fraction of a second. However, if the same server were required to serve only 50 different flows, each with a packet rate of only 1 packet/s (e.g., serving multiple handheld devices), for a total of 50 packets/s throughput, it will be unable to meet its requirements unless the same signing key and authentication tree data structure is shared across different flows or an average delay of 5 s is imposed on each flow. Such a delay is almost always unacceptable. Sharing the signing key and authentication tree data structure across several different unshared flows would result in a complex software architecture at the sender end, place unreasonable restrictions on the choice of authentication mechanisms across different flows, and expose privacy issues with regard to information being shared across different flows. The third practical problem with this scheme is that the size of the authentication information added to each message is not fixed; it depends on the short-term packet rate, which in many applications is highly irregular. During high periods, the packet overhead will be higher; this can cause additional undesirable side effects, such as increased packet loss due to fragmentation, precisely at times when the traffic volume is large.

The fourth problem is that it provides no mechanism to smooth out bursty processor loads. In any real system, there are periods when the processor has enough free time to calculate several signatures per second, and also times when the processor barely has time to calculate one or two. In this approach there is no way to leverage the idle time of the CPU to help during the times when it is highly loaded; thus, performance will seriously degrade when the CPU is heavily loaded.

The Hybrid Signature Scheme

Reference [20] proposed an approach to multicast authentication based on hybrid signatures. This approach does not suffer from any of the drawbacks of the Wong-Lam scheme described above, but requires larger-sized authentication information. The hybrid signature scheme is essentially a signature-per-packet scheme, which makes it both latency-free and immune to collusion attack. It is therefore suitable for providing strong authentication in highly interactive applications. The scheme also works (with small but significant additional size overhead) in the fully unreliable setting.

The hybrid signature solution avoids the problem of speed associated with the use of slow public key signature algorithms by using an offline/online approach similar to [21], which employs a combination of slow public key signatures with extremely fast fixed-time signature schemes based on one-way functions [22]. Standard public key digital signature schemes are slow, but a single signature key can in practice be used to sign an arbitrary number of messages. On the other hand, signature schemes based on one-way functions are extremely fast, but a single private key can only be used to securely sign a single message or a small finite number of messages [22]. In the offline/online approach, offline computation is used to create buffers of one-time key pairs and to certify the public one-time keys using the regular digital signature scheme. When a message needs to be signed, an online computation is performed to compute a signature of the message using a one-time private key from the buffer of keys, and the corresponding one-time public key and its certificate are also attached to this signature. Since operations on one-time keys are extremely fast, there is very little load or latency introduced to compute message signatures. The hybrid signature approach exploits the additional fact that generation of one-time key pairs is typically also a fast operation: it uses a technique, similar to the offline/online technique, that can sustain a high signature rate indefinitely even when both the offline and online computation is being performed at the same time by the same machine. The trick is to have the offline computation create and certify k-time key pairs instead of one-time key pairs, the most expensive of the signature creation using a regular signature scheme, can be amortized over k signatures. With this technique it is easy to see that for large values of k, the sustainable rate will approach the potential signature rate possible with the k-time key pair generation processors, and for very small values of k the rate will be close to k times the regular signature rate. By choosing an appropriate value of k, the speed of this hybrid signature scheme can be brought within the same order of magnitude as the the speed of the k-time key generation process. Reference [20] claimed speeds of 500-1000 signatures/s using this technique on workstation-class machines.

In this scheme the multicast sender has a process or thread which generates k-time key pairs and creates certificates for the k-time public keys using the sender's long-term regular signature key. The sender uses each such k-time key pair to sign k successive messages. Depending on the reliability of the network, the certificate for the corresponding k-time public key could be either sent multiple times separately or added onto each data packet or to a large fraction of the data packets. Each packet itself is signed using some ith use of a k-time signature scheme.

It is easy to see that this basic approach solves the problems associated with unreliability, multiple flows, bursty traffic, and irregular processor loading. Unreliability can be fully addressed by sending the k-time public key certificate in each packet, or more practically addressed by sending k-time key certificates multiple times within a flow to minimize the impact of a few lost packets. The offline nature of the expensive signature operation, when combined with very high throughput of the online k-time signature scheme, yields a fairly clean way to handle multiple signatures. The hybrid signature-per-packet load for each flow, buffers of k-time keys and certificates can be precomputed and filled up during periods of low CPU usage and slow traffic to tide over periods of high CPU usage and high traffic.

The major drawback of k-time one-time signature schemes that has thus far kept them from widespread use is that
time one-time public keys and/or signatures tend to be very large and thus impractical. For example, the basic hybrid approach outlined above when used in conjunction with a reasonably fast and secure A-time signature scheme would result in a size overhead of the order of 1 kbyte/packet, which is clearly impractical. However, in [20] it is shown that with the help of other cryptographic techniques, it is possible to substantially reduce the overhead to less than 300 bytes while maintaining 500-1000 signatures on workstation-class machines.

A Comparison of Techniques

Table 3 summarizes each of the authentication schemes described in this survey with respect to the evaluation criteria described earlier.

From this survey, it should be clear that currently there is no single best solution for the problem of secure authentication for multicast, but rather a range of solutions, each with its own advantages and drawbacks.

Related Work

In this survey we focus on problems and solutions related to securing multicast traffic. However, in doing so we are unable to completely cover the large body of literature and research in closely related fields such as broadcast encryption and distributed computing, both of which provide specialized techniques applicable to secure certain classes of multicast applications. In this section we briefly describe related work, and provide pointers to key publications in these and other related areas.

Broadcast Encryption

Broadcast encryption was introduced by Fiat and Naor [23] to solve the access control problem for distributing pay-per-view TV content by a broadcaster. In this setting, the broadcaster maintains a set of registered receivers and needs to distribute different programs to different subsets of subscribers from the set of registered receivers. This essentially reduces to the problem of distributing different secret keys to different sets of subscribers. Since the subsets of subscribers for different programs vary over time, the scheme needs to handle the general problem of distributing a key exclusively to an arbitrary subset of subscribers, without revealing it to receivers outside the subset. In practice, preventing all collusion-based attacks between nonsubscribers requires an exponential number of keys. This makes the scheme impractical; however, Fiat and Naor have noted that if the requirement of absolute security is relaxed to permit only a small and bounded number of colluding receivers to break the scheme, it is possible to develop schemes that are reasonable in practice. Broadcast encryption continues to be an active area of research. A related area of research is traitor tracing, which attempts to identify colluders in a broadcast encryption scheme after the scheme has been broken [24]. Another related area is the work of Abdalla, Shavitt, and Wool [25]; they relax the broadcast encryption problem to permit key distribution to a few nonsubscribers, thereby achieving even better efficiency.

Fault Tolerance and Security

A common abstraction used for designing fault-tolerant applications is that of a process group. The same concept can be extended into a security abstraction [26–30]. An integral part of this architecture is an authentication and a time service that securely and fault-tolerantly support key distribution. Together, these services are used to securely distribute group keys in support of a secure process group abstraction. Authentication of the group members and protection of group communication is achieved through the use of the group keys. One of the main consequences of using secure group abstractions on fault tolerance is that the group abstraction is itself virtually synchronized. This results in a toolkit that can be used for developing highly secure and fault-tolerant applications. As a result, the protocols tend to be complex and expensive.

Enclaves

Gong describes an approach for integrating secure protocols into user-level group-oriented applications. The Enclaves toolkit [31] achieves this by providing an application programming interface (API) for user authentication, key distribution, secure group management, and secure multicast among group members. This API can be used by application builders for secure group communication. The Enclaves toolkit runs completely in user space and does not need infrastructure support such as a multicast kernel. Communication is built on top of TCP/IP. The toolkit assumes a centralized architecture consisting of a group leader and members. The group leader and members securely authenticate themselves to each other via authentication tokens. Once this is done, a session key is securely distributed to the new members, and securely revoked and updated whenever a member leaves the session. Group communication is secured by encrypting network traffic. For secured multicast, for each group session different keys are distributed to members for encrypting multicast data. The Enclaves toolkit does not specifically address the issues of efficient key management and authentication.

Extensions of Diffie-Hellman Key Exchange Protocols

Considerable research has been done on extending the two-party Diffie-Hellman key agreement protocol to groups. In contrast to the key distribution protocols covered in this survey, these protocols focus on the issue of key agreement. These protocols are contributory, in that they allow a shared key to be derived by two (or more) parties as a function of information contributed by each party. Several articles have appeared on this subject [32–36]. One drawback of these schemes is that they don't scale well; the amount of information exchanged during the process of key establishment is proportional to the size of the group.

Policy Issues

The increasing use of multicast as a fundamental building block for a variety of distributed applications has focused
attention on the different and varying levels of service required by different users. While a lot of work has been done on security mechanisms (e.g., the key management and authentication techniques described earlier), relatively less attention has been paid to the issue of policies for secure network communication [4, 30]. One such effect, Antigone [4], provides a suite of basic policy components that address issues such as session reserving, application messages, group membership, and process failure. These components can be composed and configured to develop flexible application security policies. Developers may either choose a policy from a set of standard policies or design a custom policy that best suits their security and performance requirements.

Conclusion

In this survey article, we discuss the characteristics of the secure multicast problem that make it a challenging research issue. We present a list of evaluation criteria for secure multicast solutions, and review important work in the areas of secure multicast group architecture, group key management, and packet source authentication. None of the work we review can effectively solve the secure multicast problem by itself; an ideal solution would choose the best components from each of the solutions outlined and combine them as appropriate. Furthermore, this research area is still relatively young, and new techniques will surely evolve over time to replace or supplement those that currently exist.

References


Additional Reading


Biographies

MATTHEW J. MOYER (mjy@rice.edu) is a doctoral student in the College of Computing at Georgia Tech. His research interests include computer systems security and information security. He is currently working on the design of a framework for more flexible context- and state-based access control to data. He is also interested in proactively securing systems and secure, scalable multicast communications. He has received an M.S. degree in computer science from Georgia Tech (1999), a B.S. degree summa cum laude in mathematics from Bucknell University (1997), and a B.A. degree in computer science from Bucknell University (1999). He is currently working as a graduate research assistant at Georgia Tech, and worked previously as an intern at IBM’s T.J. Watson Research Center (1998), a graduate teaching assistant at Georgia Tech (1997, 1998, 1999), a campus computing liaison of Bucknell University (1997), and an intern at KPMG Peat Marwick (1995) and Arthur Andersen (1994). He received the award for Outstanding Student in Computer Science from Bucknell’s College of Arts and Sciences (1997) and the Miller Prize for Outstanding Thesis at Bucknell University (1997). He is a member of ACM, USENIX, the Phi Mu Epsilon mathematics honor society, and Phi Beta Kappa.

JOSHUA R. RICH (jrich@watson.ibm.com) manages the Internet Security and Technology Group at IBM’s Thomas J. Watson Research Center. He obtained his doctoral degree from the University of Texas at Austin in 1992. His research interests include programming methodology, distributed systems, network security, and cryptography. He is a member of IFS’s Working Group 2.1 on Programming Methodology.

PAULUS ROBERTS (robertsp@watson.ibm.com) is a research staff member of IBM’s Thomas J. Watson Research Center. He received his Ph.D. from Cornell University in January 1994. His research interests include applied cryptography, network and systems security, complexity theory, and randomized algorithms.