Network Security in the Multicast Framework

Refik Molva, Alain Pannetrat

Institut EURECOM
2229 Route des Crêtes
F-06904 Sophia Antipolis Cedex, FRANCE
{Refik.Molva, Alain.Pannetrat}@eurecom.fr
http:www.eurecom.fr/~nsteam

Abstract. This chapter is an overview of security requirements raised by multicast communications along with a survey of principal solutions. A detailed requirement analysis points out the characteristics of multicast in terms of security and scalability. The differences between unicast and multicast communications with respect to the design of authentication and confidentiality mechanisms are discussed. Main confidentiality proposals are analyzed in a detailed comparison survey based on the security and scalability criteria.

1 Introduction

This chapter addresses network security in the framework of multicast communications. Both *security* and *multicast* encompass a broad number of problems. We have chosen to deal with two most fundamental aspects of security in the context of multicast: *confidentiality* and *authentication* services for multicast applications. We start with an overview of IP-Multicast and we highlight the main security requirements of large scale applications which are build upon multicast mechanisms.

IP-Multicast was introduced by STEVE DEERING [Dee89,Dee91] and describes a set of mechanisms which allow the delivery of a packet to a *set* of recipients rather than just one recipient as opposed to *unicast* communications. The set of recipients that receive the same multicast packets are considered members of a *multicast group*. In the multicast setting, the sender transmits a single copy of a packet and the network takes care of duplicating the packet at proper branching points in order for each recipient to receive a copy of the original packet. With this approach, only one copy of the original packet will be transmitted on most of the underlying network links, which saves resources compared to an equivalent set of unicast communications, as illustrated in the example of figure 1.

2 The Security of Multicast Applications

The integrity and accuracy of multicast routing information is important for the proper functioning of IP-Multicast. Adversaries may inject bogus routing messages in the network to modify or even disrupt the routing mechanisms. This type of exposure on routing mechanisms however is not particularly different in multicast. There exist efforts to secure the multicast routing protocols themselves [Moy94,Fen97] and we will not further address this issue in this chapter. The focus of this chapter is geared towards the security of applications which are built on top of IP-Multicast, regardless of lower layer security requirements.

Many large scale commercial applications such as video/audio broadcasting or stock quote distribution could benefit from IP-Multicast mechanisms to reach many receivers in a scalable way. In fact most prominent multicast applications have been confined to the enterprise level or for non-commercial uses such as IETF or NASA broadcastings[MS97]. One of the factors behind the reluctance to the large scale deployment of commercial applications

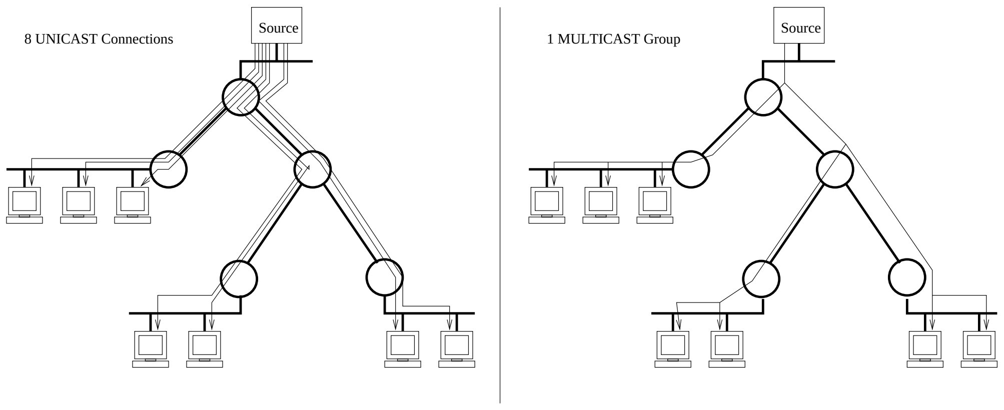

Fig. 1. Unicast or Multicast

using multicast may well be security. Indeed, securing multicast applications turns out to be a challenge in many situations, mainly because existing unicast security solutions cannot be extended to the multicast setting.

2.1 An application scenario

One of the best ways to illustrate the difficulty of securing multicast applications is to use a simple application scenario as an illustration.

Scenario Description. Consider for example a large financial news broadcaster who wants to create a business by providing financial data and market analysis to a large set of customers on the Internet through multicast. These customers are expected to pay for accessing the broadcast content for variable lengths of time, ranging from less than an hour to several days, depending on their needs.

Security Issues. This financial news broadcaster is faced with two main security issues related to the distribution of its commercial content over the Internet:

Confidentiality: the content should only be accessible to the clients who payed for the service, and only for the duration corresponding to the payment.

Authentication: the content provider needs to prevent its impersonation by another entity who could try to generate content on its behalf, possibly harming its image and credibility.

Adapting Unicast Solutions In unicast communications, two efficient techniques are frequently used to address these issues: symmetric encryption[BDJR97] and MACs (Message Authentications Codes)[BCK96]. To refer to these mechanisms informally, we will denote the symmetric encryption of a plaintext message M with a key K as $C \leftarrow \mathcal{E}_K(M)$ and the corresponding decryption as $M \leftarrow \mathcal{D}_K(C)$. We will write $\sigma \leftarrow \mathcal{T}_{K^{mac}}(M)$ the algorithm which takes a message M and a MAC key K^{mac} to produce a MAC tag σ and we denote $\{1,0\} \leftarrow \mathcal{V}_{K^{mac}}(M,\sigma)$ the corresponding verification algorithm which returns 1 if $\sigma = \mathcal{T}_{K^{mac}}(M)$ and 0 otherwise. Informally speaking, it should be computationally infeasible for an adversary who sees m pairs $\{M_i,\sigma_i\}$ to generate a new pair $\{M',\sigma'\}$ which verifies $\mathcal{V}_{K^{mac}}(M',\sigma')$ such that $\{M',\sigma'\} \neq \{M_i,\sigma_i\}$ for $i = 1,...,m$ without the knowledge of K^{mac}.

Confidentiality. If the multicast data is encrypted with a single key K common to all recipients who paid for the service then it should prevent others from accessing the data. However, when the subscription time of one of the clients $R_j$expires then K must be updated with a new value K' in order to prevent the client R_j from further accessing the data. Without loss of generality, we can assume in our scenario that each client R_i shares a private encryption key K_i with the source. If K is the only secret shared between the source and the recipients, in our setting there is no simple way to exclude a client from the recipient group other than sending: $\mathcal{E}_{K_i}(K')$ for all $i \neq j$. This approach is clearly unscalable in anything but relatively small groups.

Restricting access to the multicast data through the use of encryption poses another more subtle problem: containing security exposures. Indeed, in a large group there is a non negligible chance that one recipient will be compromised and that its keys will be exposed. Let us suppose for example that the key K we used above to encrypt data is exposed on a web page or in a public newsgroup; this allows anyone within the scope of the multicast group to access the data. Unlike in unicast communications, the adversary does not need to be connected to the link or to corrupt the routing protocol between the communication endpoints, since multicast data is forwarded automatically to him if he requests to join the group. Consequently, we need to provide mechanisms that limit the impact of such exposures.

Authentication. In the multicast setting, if we construct a Message Authentication Code or MAC[BCK96] for each packet with a common MAC key K^{mac} shared between the recipient clients and the source, it will disallow non-clients who do not possess K^{mac} to forge packets that will appear to originate from the source. Since a MAC is computed efficiently using symmetric cryptographic techniques, each packet can be verified independently, thus, lost packets will not affect the ability to authenticate others. But this approach has one big drawback: any client can impersonate as the source, by generating a message M and a valid tag $\sigma = \mathcal{T}_{K^{mac}}(M)$ from the key K^{mac}. Since the MAC key K^{mac} is common to all clients and the source, there is no way to distinguish a packet generated by the source from a packet generated by a client.

An alternative is to have a different key K_i^{mac} for each recipient R_i and append a list of the corresponding MACs to each packet as:

$$M, \sigma_1 = \mathcal{T}_{K_1^{mac}}(M), \sigma_2 = \mathcal{T}_{K_2^{mac}}(M), ..., \sigma_n = \mathcal{T}_{K_n^{mac}}(M)$$

This approach is clearly impractical in a large group since the overhead per packet will increase linearly with the group size.

A third alternative is to replace the MAC with a digital signature (Digital signatures also provide non-repudiation of origin which was not a requirement in our scenario). However, this solution also has its drawbacks because digital signatures are based on asymmetric techniques that introduce a significantly higher computational cost and a non negligible communication overhead, making this alternative impractical in most scenarios.

A Clash Between Security and Scalability

Though the previous example is somewhat simplified, it highlights some of the main difficulties associated with the design of security protocols for multicast in large groups. There seems to be an inherent clash between multicast and security. In the case of confidentiality, there is a conflict between multicast scalability mechanisms and security. Indeed, multicast relies on a single group address to identify the set of recipient rather than explicitly listing them. This anonymous identification is the primary mechanism which allows multicast to scale to virtually any group size. On the other hand, confidentiality requires us to identify explicitly the entities which send or receive data in order to provide them with the right keys to access the encrypted multicast data. In the case of authentication, the problem is not related to the group size explicitly but rather to the requirement of an efficient asymmetric mechanism to disallow receivers from impersonating the sender. Additionally, since most multicast protocols are implemented over a best effort channel, these authentication mechanisms need to tolerate losses.

3 Multicast Confidentiality

The sequel of the chapter focuses on multicast confidentiality. We first present a detailed analysis of the problem that helps us draw the main requirements for multicast confidentiality in large groups in terms of security and scalability. Based on these requirements we then present an overview of existing multicast confidentiality proposals.

To clarify our discussion of encryption in the context of multicast we will use the following naming conventions:

Source: We call *source* an entity that wishes to transmit a certain *content* to a selected set of chosen recipients using multicast.

Content: We always use the words *multicast content* to refer to the actual cleartext data that the source wishes to transmit securely over a multicast channel.

Recipient: We call *recipient* any entity capable of receiving multicast packets from a certain group, regardless of its capacity to actually decrypt the multicast packets to access the content. The only limit to the set of recipients is dictated by multicast routing protocol restrictions, mainly the scope of the multicast group which is limited by the TTL selected by the source.

Member: We call *member*, a selected recipient which has been given cryptographic keys that enable it to access the *content* of the received multicast packets.

Membership Manager: A *membership manager* describes the entity (or entities) which grants or refuses membership to recipients.

The main focus of this section is to analyze and propose methods that allow a dynamic set of members to access the multicast content transmitted by a source while disallowing other recipients to do so.

Typical large scale multicast application scenarios which require confidentiality services are:

- Pay-per-view TV,
- High quality streaming audio,
- The distribution of software updates,
- News feeds and stock quote distribution,

All these scenarios involve one of very few sources and potentially a very large amount of members or clients. In this paper we have chosen to put an emphasis on solutions geared towards these scenarios. In particular we do not deal with dynamic peer groups [BD95,STW96,STW98] which involves secure *n-to-n* communications in much smaller groups through the cooperation of the entities in the group. Though this work deals with a selective broadcasting medium, we do not either aim to provide a solution to the problem of *broadcast encryption* [FN93] where each message is encrypted for an arbitrary and potentially disjoint subset of a larger known group. In particular, in our setting, the set of potential members is not necessarily known a priori. Consequently, we assume at least a minimal unicast back channel between the recipients and a membership manager. This channel is used at least once when a recipient sends a request to a membership manager to become a member of the group.

4 The Security of Dynamic Groups

We qualify the set of members as *dynamic* because we consider the general case where the set of members evolves through time as members are *added* or *removed* from the group during a session. We define *add* and *remove* operations as follows:

Add: When a *recipient* becomes a *member* of the secure multicast group by receiving proper cryptographic access parameters, we say that he is *added* to the group. To be added to a group, recipients needs to contact a membership manager through a secure authenticated unicast channel. If the recipient is allowed to become a member then it receives keys and other related parameters necessary to start accessing the multicast content. The membership policy is application dependent and may be subordinated to other issues such as payment and billing, which are all beyond the scope of this chapter.

Remove: We say that a *member* is removed from the group, when its ability to access the encrypted multicast content is suppressed by the membership manager. A membership manager can decide to remove a member from the group at any time during the session or at least on a certain interval boundary that is application dependent (a packet, a frame, a quantity of bytes or time...).

In many publications, add and remove operations are referred as *join* and *leave* operations, respectively. We prefer the former terminology because it highlights more clearly the fact that providing access to the multicast content is ultimately the decision of a membership manager rather than the recipient itself. We will restrict the terms *join* and *leave* for multicast routing, to describe the fact that an entity requests or relinquishes receiving multicast packets, independently of any security concerns.

All the application scenarios we described in the previous section involve dynamic groups where members are added or removed from the member group. A dynamic set of members implies two security requirements that are not found in traditional secure unicast communications:

backward secrecy and *forward secrecy.*

Backward secrecy defines the requirement that an added member should be able to access multicast content transmitted before it became a member. This means that if a recipient records encrypted multicast data send prior to the time it is added to the group, it should not be able to decrypt it once it becomes a member. Similarly, *forward secrecy*[1] defines the requirement that a member leaving the group should not be able to further access multicast content.

5 Algorithmic Requirements

The forward and backward secrecy requirements have a immediate consequence: the parameters of the encryption algorithm that protects the multicast content must be changed each time a member is added or removed from the group. We must simultaneously allow members to infer the new parameters used to access the multicast data while disallowing other recipients to do so. This parameter change must be done in a scalable way, independently of the group size, which can be a real challenge, as illustrated by the simple financial content provider scenario in section 2.1. The scalability issues related to multicast key management in dynamic groups were first highlighted in the IOLUS[Mit97] framework by S. Mittra, who identified two generic problems:

[1] The term "forward secrecy" is used here in the context of multicast security and should not be confused with *perfect forward secrecy* in unicast communication, which deals with the security of past sessions when a session key is compromised.

1. The "one does not equal n" failure which occurs when the group cannot be treated as a whole but instead as a set of individuals with distinct demands.
2. The "one affects all" failure which occurs when the action of a member affect the whole group.

Our experience with many multicast proposals has prompted us to refine these definitions to propose the following two requirements:

Processing scalability: The cost supported by an individual component, be it the source, an intermediary forwarding component or a member, should scale to any potential group size and membership duration.

Membership robustness: Members should be able to access the content of received multicast packets *as soon* as they are received.

The *processing scalability* requirement was informally illustrated by our sample scenario in section 2.1, where we tried to use unicast techniques in a multicast setting to provide confidentiality. To remove a member from a group of N members it required the source (or the membership manager) to send out a new global key encrypted with $(N-1)$ different private keys to replace the previous global key used to access the content in order to assure forward secrecy. Clearly, this method does not offer processing scalability.

The best effort nature of the Internet is a potential factor that greatly affects membership robustness. Consider a scenario in which the membership manager is required to broadcast a single value R to the whole group each time a member is removed or added, in order for members to update their decryption parameters to continue accessing the multicast content. Several multicast applications can tolerate packet losses. However, if some members experience the loss or the delay of R they will not be able to update the keys needed to access the multicast content, thus the newly received content packets will be worthless. Consequently, in such a scenario, add and remove operations may impair membership robustness.

6 Collusion, Containment and Trust.

While the main goal of multicast encryption in a dynamic group is to come up with a protocol which satisfies both the security requirements and the algorithmic requirements, this is still not quite sufficient to provide security in a *large* group. A factor that is often overlooked in most secure multicast proposals is the potential compromise of a member. Indeed, as the member group becomes larger, the probability of member key exposure increases. In a sufficiently large group, there is no doubt that an exposure will occur. These exposures may be done intentionally by a member: it could send its security parameters to another recipient, or they may be unintentional: a "hacker" may steal the parameters held by a member and publish them in a newsgroup or on a web page. Thus, one of the important concepts we put forward in this chapter is that, since we cannot avoid security exposures, we need to limit the impact of such exposures, a property that we call *containment*.

A second issue is *collusion*: if a few members collude together they should not be able to elevate their privileges beyond the sum of their own privileges. For example, consider a scenario where each member holds secret keys that allow them temporary access to the group. If these members are able to achieve unlimited access by exchanging their secrets and making some computations, then the scheme is clearly weak and will be subverted quickly in a large group.

Some multicast schemes use intermediate elements in the network, such as routers, subgroup servers or proxies as participants in the security protocol. These elements are external to the member group and most likely not always fully controlled by the content provider, the members or a membership manager. In fact, it is quite possible that these intermediary elements will be shared between several secure multicast groups with different member sets.

Moreover, in some situations, these elements are numerous enough such that the compromise of one or a few of these entities is probable. Consequently, while it is quite reasonable to assume that the source as well as the few other entities such as a membership manager are secure, this is not realistic for other network resources. Thus we need to *limit the trust* placed in these intermediate elements if they are involved used in security protocols.

7 Requirement Summary

Confidentiality brings out significantly more requirements in the multicast setting than in the unicast setting. We summarize these requirements here as a reference for the analysis of current multicast proposals in the following sections.

Security Requirements:

Requirement 1 Data Confidentiality: *the multicast content should be only accessible to members.*

Requirement 2 Backward and Forward Secrecy: *A new (resp. past) member should not have access to past (resp. future) data exchanged by the group members.*

Requirement 3 Collusion Resistance: *A set of members which exchange their secrets cannot gain additional privileges.*

Requirement 4 Containment: *The compromise of one member should not cause the compromise of the entire group.*

Requirement 5 Limited Intermediary Trust: *Intermediary elements in the network should not be trusted with the security of the group.*

Strictly speaking, the latter requirement (5) is only relevant to protocols which use intermediary elements to actively participate in the security of the group.

Algorithmic requirements:

Requirement 6 Processing Scalability: *The cost supported by an individual component, be it the source, an intermediary forwarding component or a member, should scale to any potential group size and membership duration.*

Requirement 7 Membership Robustness: *Members should be able to access the content of received multicast packets as soon as they are received.*

8 Overview of Multicast Confidentiality Algorithms

This part of the chapter includes the description of three different multicast confidentiality algorithms: the Logical Key Hierarchy, the re-encryption trees of IOLUS, and the MARKS[Bri99] approach. In the sequel of this chapter, we denote $E_K(x)$ as the encryption of message x with key K with a standard symmetric encryption technique[BDJR97,oST01].

9 LKH: The Logical Key Hierarchy

The Logical Key Hierarchy was independently proposed by Wong et al. [WGL98] and Wallner et al.[WHA98].

9.1 Construction

Consider a set of n members $\{R_1, ..., R_n\}$. We build a balanced (or almost balanced) binary tree with n leafs, with a one to one correspondence between a leaf L_i and a member R_i. A random key k_j is then attributed to each vertex j in the tree, including the root and the leafs themselves. This construction that we will call *logical key hierarchy* (LKH) is is illustrated on figure 2 with a small set of 7 recipients.

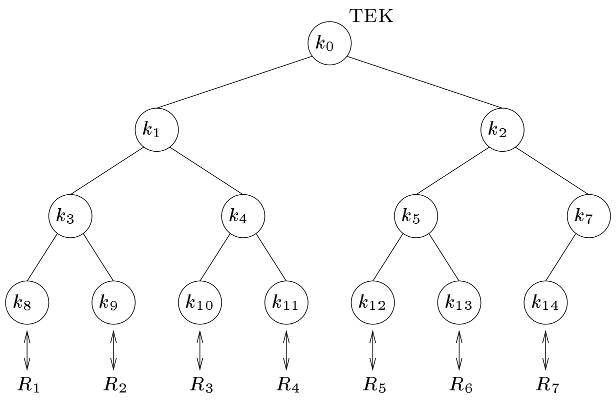

Fig. 2. Basic key graph with 7 members

Setup. Each member R_i receives the set of keys corresponding to the path from the root of the tree to its corresponding leaf L_i. The root of the tree which is known by all members is called the *Traffic Encryption Key* (TEK) and represents the symmetric encryption key used to protect the content distributed over the multicast channel. The remaining keys in the tree are often called *Key Encryption Keys* and are used to update the TEK to guaranty backward and forward secrecy. Referring to our example in figure 2, member R_1 would receive the key set $\{k_0, k_1, k_3, k_8\}$ where k_0 represents the TEK and $\{k_1, k_3, k_8\}$ the KEKs.

9.2 Usage

Let us denote $\mathcal{L}(k_i)$ (respectively $\mathcal{R}(k_i)$) the predicate which returns *true* if the node representing key k_i has a left child (respectively a right child). Further more we will denote $\mathcal{LK}(k_i)$ the key held in the left child of the node representing k_i if it exists, and we similarly define $\mathcal{RK}(k_i)$ for the right child.

To *remove* a member R_i from the group: All keys representing the path from the root to the the leaf L_i corresponding to departing member R_i are invalidated. The leaf L_i corresponding to the departing member is removed from the tree. All the remaining invalidated keys $\{k_1, ..., k_l\}$ in the path from the root to the former leaf are replaced with new random values $\{k'_1, ..., k'_l\}$. For convenience consider $\{k'_1, ..., k'_l\}$ to be ordered by *decreasing* depth in the tree. To allow the remaining members in the tree to update their keys, the membership manager proceeds as follows:

Algorithm 11 *LKH update*

```
For $i = 1, ..., (l-1)$ do
    if $\mathcal{L}(k'_i)$ then multicast $E_{\mathcal{LK}(k'_i)}(k'_{(i-1)})$
    if $\mathcal{R}(k'_i)$ then multicast $E_{\mathcal{RK}(k'_i)}(k'_{(i-1)})$
```

None of the keys that are used for encryption in the previous algorithm are known by the removed member. However, all the remaining members will be able to recover enough keys to update their own key set.

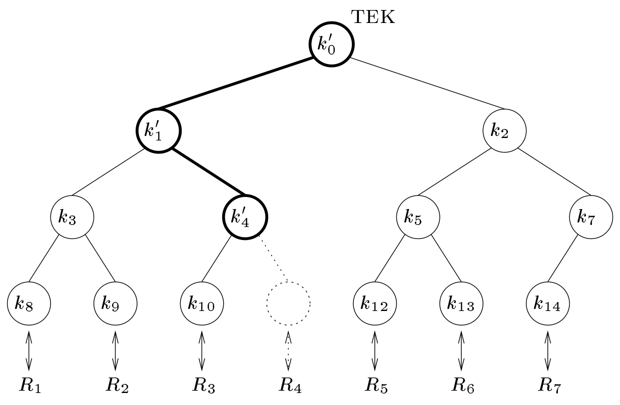

Fig. 3. LKH: A member leaving the group.

As illustrated on figure 3, if member R_4 leaves, then the membership manager needs to broadcast $E_{k_{10}}(k'_4)$, $E_{k_3}(k'_1)$, $E_{k'_4}(k'_1)$, $E_{k'_1}(k'_0)$ and $k_{k_2}(k'_0)$.

To *add* a member to the group: When a member is added to the group, the tree is extended with an additional node. If all leafs in the tree are already attributed we can take a leaf L_i and create two children: the left child is assigned to the member associated with the former leaf L_i and the right child is assigned to the newly added member. Alternatively, we can simply design the tree to be deep enough to accommodate the potential maximum number of members.

Assuming that the new member corresponds to leaf L_i, the sender makes all the keys in the nodes on the path from L_i to the root invalid. A random key is assigned to L_i and transmitted to the added member with a secure unicast channel. All other nodes in the path are updated with the same algorithm that was used above to remove a member from the group.

Key Usage Summary. The scheme distinguishes two type of keys: a TEK and KEK. The TEK is the encryption key that protects the multicast content. It is changed each time a member is added or removed from the group. The KEKs are used to distribute and update the TEK for all members. Until a valid KEK set is explicitly invalidated by the membership manager, it can be maintained in a valid state provided that all update messages are received. Consequently a valid KEK set represents membership in a group and provides long term access to the group, while the TEK itself only represents a short term access to the content.

9.3 Algorithmic Properties of the Scheme

Considering a set of at most n members, the basic LKH scheme requires each recipient to store $\lceil \log_2(n) \rceil + 1$ keys. Each addition or removal of a member requires the broadcast of $2. \lceil \log_2(n) \rceil - 1$ messages (alternatively a single message with all key update messages grouped together). Thus, the processing load of the components and the key message sizes increase logarithmically with the group size, which allows to achieve good processing scalability (req. 6).

However, if a member fails to receive just a single key update message because of a network loss, then it cannot decrypt the new content. Losses have a worsened impact because subsequent key updates will often depend on previous ones. Consequently a member cannot always resynchronizes its parameters on a new key update message if it lost one of the previous key update messages. A member who didn't receive a KEK update message might have to re-register with the membership manager through an authenticated secure unicast channel to receive the proper keys. If many members do this at the same time we may run into scalability issues. Consequently in this basic version of the scheme, both add and remove operations do not provide membership robustness (req. 7) in the presence of losses or strong key update delays.

9.4 Improvements

The basic scheme we described previously has been constantly improved by several authors. We summarize these contributions here.

No-message Member Addition An interesting improvement was proposed by CARONNI ET AL. [CWSP98] which eliminates the need to multicast a key update message when a member is added to the group. In their proposal, the set $\{k_1, ..., k_l\}$ of invalidated keys represented on the path from the root of the tree are replaced with a set $\{k'_1, ..., k'_l\}$ where each key k'_i is the result of a one way function applied to the previous key, that is $k'_i = F(k_i)$ where F is a publicly known one-way function. This allows members to compute the new keys from the previous ones, and the one-way property of F disallows the new member to compute the previously used keys. The set $\{k'_1, ..., k'_l\}$ is transmitted to the newly added member through a secure unicast channel. In their work CARONNI ET AL. use a counter (a sub-version number) for the TEK and KEKs as ancillary information in the multicast data packets to keep track of the number of times the one-way function was applied (this naturally adds a few bytes of overhead to each packet). Since adding a member does not require the transmission of a multicast message anymore, membership robustness (req. 7) is always assured during this operation, and the problem is now restricted to the removal of a member.

Halving the Update Overhead MCGREW AND SHERMAN were the first to propose a method to halve the communication overhead of add and remove operations, with a construction called OFT (One Way Function Trees, [MS98]). This further increases the processing scalability of the protocol. We will not describe the OFT construction here, instead we will focus on the more recent ELK approach presented below, which shares some similarity with OFT.

More recently CANNETI ET AL. have proposed a method[CGI+99] that achieves the same reduction and that is compatible with the "no-message-member-addition" we described above. We will describe the core idea behind their protocol here. Assume that f is a one way function, and let $f^k(x)$ denote k successive applications of f to x, where $f^0(x) = x$. As usual, when a member is removed from the group, the remaining keys $\{k_1, ..., k_l\}$ represented on the path from the root to the departing leaf need to be changed to a new set $\{k'_1, ..., k'_l\}$. Here, instead of choosing all these keys at random, the membership manager chooses only one random value r and sets

$$k'_l \leftarrow r,\ k'_{(l-1)} \leftarrow f^1(r),\ k'_{(l-2)} \leftarrow f^2(r),\ ...,\ k'_1 \leftarrow f^{(l-1)}(r)$$

The membership manager then encrypts each k'_i with the key represented by the child node of k'_i that is **not** in the invalidated path. Thus each individual k'_i is encrypted only once as opposed to the original LKH protocol in algorithm 11 which encrypted each key with both child node keys. Because of the relationship between the keys created by f, the nodes can still reconstruct the set of keys they need by applying f.

Example: Recall figure 3 in which member R_4 is removed from the group, the membership manager will compute a random value r and send $E_{k_{10}}(r)$, $E_{k_3}(f(r))$, $E_{k_2}(f(f(r)))$.

- Member R_3 decrypt $k'_{10} = r$ and computes $k'_4 = f(k'_{10})$, $k'_1 = f(f(k'_{10}))$ as well as $k'_0 = f(f(f(k'_{10})))$.
- Members R_1 and R_2 decrypt k'_1 and compute $k'_0 = f(k'_1)$.
- Members R_5, R_6, R_7 decrypt k'_0.

Assuming that the tree is balanced, the communication overhead is at most $\lceil \log_2(n) \rceil$ keys instead of $\lceil 2.\log_2(n) - 1 \rceil$ keys as in the original LKH construction.

Increasing Reliability Clearly, if the network was perfectly reliable over the whole multicast group the removal of a member would not introduce any membership robustness issues in the LKH scheme (req. 7). However, multicast uses an unreliable transport protocol such as UDP. Consequently, increasing the reliability of the leave operation is crucial if we want to reduce membership robustness issues (req. 7).

Wong and Lam implemented their LKH[WGL98] protocol over IP-Multicast in Keystone[WL00], and use FEC (Forward Error Correction) to increase the reliability of the addition and removal operations (since they do not use the no-message-member-addition method). Recall that when a member is added or removed from the group in the basic scheme, a set of keys $\{k'_1, k'_2, ..., k'_l\}$ is encrypted with other keys in the tree and transmitted to the group. In the Keystone scheme, these encrypted keys are represented over s packets to which we add r parity packets. The receiver may reconstruct the encrypted key set if he recovers any subset of s packets out of the $(s+r)$ transmitted packets. Since we do not have an absolute guaranty that s packets out of $(s+r)$ packets will always arrive, they combine this mechanism with a unicast re-synchronization mechanism that allows a member to contact a "key cache" server in order to update its keys if losses exceed the correction capacity of the FEC. In their work, they consider a 10-20% loss rate and they show that adding redundancy can substantially increase the likelihood that a member will be able to update its keys. However, as noted by Perrig et al.[PST01] in criticism of the Keystone scheme, the authors assume independent packet losses to compute the key recovery probabilities, while it has been shown that Internet losses occur in bursts.

Perrig et al. proposed a protocol called ELK[PST01] which combines previously proposed ideas such as "no message member addition" and halving the update overhead, with new tricks to increase the reliability of the KEK update operation. The main idea of their work stands in two points:

- First, send a key update message U with reduced communication overhead.
- Second, send many small *hints* that allow a member to recover updated keys if U is lost. The *hints* are smaller than U but require a much higher computational cost from the recipient.

We will give a slightly simplified description of their protocol, omitting some cryptographic key derivation issues that are described in detail in their work[PST01] but are not useful to understand the methods employed here.

Reducing the communication overhead. To reduce the communication overhead of a key update they construct new keys based on contributions from the two children nodes. Consider for example a node containing the key k_i on the path from the root to a leaf representing a removed member which needs to be updated to a new value k'_i. Let k_R (*resp.* k_L) define the key held by the right (*resp. left*) child of the node associated to k_i. Denote $F_k^{\langle p \to m \rangle}(x)$

as a pseudo-random function which takes a p bit value x as input and outputs m bits using a key k. In the ELK construction, the keys in a node are p bits long and are derived from p_1 bits of the left child and p_2 from the right child. To update k_i to k'_i we first derive the right and left contributions C_R and C_L from the old key k_i as follows:

$$C_L \leftarrow F_{k_L}^{\langle p \rightarrow p_1 \rangle}(k_i)$$

$$C_R \leftarrow F_{k_R}^{\langle p \rightarrow p_2 \rangle}(k_i)$$

Now we compute the new key k'_i by assembling these two contributions to derive k'_i:

$$k'_i \leftarrow F_{(C_L||C_R)}^{\langle p \rightarrow p \rangle}(k_i)$$

Consequently, to derive k'_i from k_i the members find themselves in two possible situations:

- They know k_L and thus C_L, and they need to receive C_R to compute k'_i, *or,*
- They know k_R and thus C_R, and they need to receive C_L to compute k'_i.

Consequently, the membership manager will send $E_{k_L}(C_R)$ and $E_{k_R}(C_L)$ to the group where E denotes a stream cipher rather than a block cipher (using the old root key or TEK as Initial Vector). Using a stream cipher allows the membership manager to send $\langle E_{k_L}(C_R), E_{k_R}(C_L)\rangle$ over exactly $p_1 + p_2$ bits. A full key update will thus require the broadcast of $log_2(n)$ chunks of $p_1 + p_2$ bits. Considering $p_1 + p_2 \leq p$ this amounts to having at most 50% of the communication overhead of the original LKH protocol.

Using Hints to increase reliability. Consider the case where $p_1 < p_2$ and where in fact p_1 is small enough such that computing 2^{p_1} symmetric cryptographic computations remains feasible in a reasonably short time. In this situation it's possible to construct an even smaller key update procedure that the authors of ELK call a *hint,* which is defined as the pair $\{V_{k'_i}, E_{k_L}(h)\}$ where $V_{k'_i}$ is a p_3 bit cryptographic checksum computed as follows :

$$V'_i \leftarrow F_{k'}^{\langle p \rightarrow p_3 \rangle}(0)$$

With $LSB^{\langle p_2 - p_1 \rangle}(x)$ being the function which returns the $(p_2 - p_1)$ least significant bits of x, h is defined as:

$$h \leftarrow LSB^{\langle p_2 - p_1 \rangle}(F_{k_R}^{\langle n \rightarrow p_2 \rangle}(k_i)) = LSB^{\langle p_2 - p_1 \rangle}(C_R)$$

The hint $\{V_{k'_i}, E_{k_L}(h)\}$ of which the second member h is encrypted with k_L, is broadcasted to the group after the initial key update we described previously. The hint $\{V_{k'_i}, E_{K_L}(h)\}$ can be used to recover the value k'_i in a node. Again, depending on left or right considerations, there are two ways to use the hint:

- Members who know k_L can decrypt h, thus they can recover the $(p_2 - p_1)$ least significant bits of C_R. Since they also know C_L they have to make and exhaustive search for the n_1 missing bits of C_R to recover a value of the form $C_L||\widetilde{C}_R$. For each candidate $C_L||\widetilde{C}_R$ they compute the corresponding $\widetilde{k}'_i$ as $\widetilde{k}'_i \leftarrow F_{(C_L||\widetilde{C}_R)}^{\langle p \rightarrow p \rangle}(k_i)$ and use the checksum $V_{k'_i}$ to verify that the candidate is good, otherwise they continue the exhaustive search.
- Members who know k_R will perform an exhaustive search through the 2^{n_1} values that k_L can take. For each potential k_L value they compute the corresponding candidate for k'_i. They use the checksum $V_{k'_i}$ to validate the right candidate.

In practice the authors of ELK suggested in a example to use the values $p_1 = 16$, $p_2 = 35$, $p_3 = 17$ and $n = 64$ using the RC5 cipher[Riv95] both for encryption and also to construct a CBC-MAC based pseudo random function. The authors of ELK implemented this example on a 800Mhz Pentium machine to achieve a high number of computational operations per second.

Aggregating Member Additions and Removals. It should be clear from the description of the LKH algorithm that frequent membership changes increase the workload of all the members of the group and require the membership manager to broadcast a proportional number of key update messages. In situations where the group has a highly dynamic set of member, the membership manager will have to deal with both membership changes and re-synchronization messages for recipients who failed to update their keys correctly, which creates a potential bottleneck. Consequently several authors [CEK$^+$99,LYGL01] have proposed methods to reduce this problem by combining several removal operations into one to achieve a lower communication overhead than the sum of these individual removal operations. In a related but more theoretical approach, [PB99] proposes to organize members in the tree according to their removal probability to increase the efficiency of batch removals and additions.

9.5 LKH: Summary and Conclusion

The LKH construction is an interesting approach, which satisfies the following requirements of multicast confidentiality:

– *Data confidentiality* (req. 1).
– *Backward and forward secrecy* (req. 2).
– *Collusion Resistance*[2] (req. 3).
– *Processing scalability* (req. 6).

Beyond classical multicast forwarding mechanisms, it does not require any additional contribution from the intermediary elements in the network, which clearly also satisfies requirement 5. While some authors have tried to increase the processing scalability of the scheme, the main concern about LKH scheme is membership robustness. Indeed, each time a member is added or removed from the group, all the members need to receive a key update message. If that message is not received by some members, it will disallow them to access the multicast content and it may also disallow them to access further key update messages. The Keystone framework [WL00] uses FEC to increase the likelihood that the key update message will be received by the members, while the ELK framework relies on small "hints" that can be used by a member to derive missing keys in combination with computational tradeoffs.

There remains one requirements that none of the LKH protocol addresses: Containment (req. 4). Indeed, each set of keys held by a member can be used anywhere in the network within the scope of the multicast group to access the encrypted content.

10 Re-Encryption Trees

In 1997, Mittra proposed the Iolus framework, which partitions the group of members into many subsets and creates a tree hierarchy between them. Each subset of members is managed by an distinct entity called a Group Security Intermediary (GSI). The central or top-level subgroup in the hierarchy contains the central group manager called Group Security Center (GSC), as illustrated on the example on figure 4.

10.1 Basic scheme

The GSC produces data encrypted with a key K_1 and multicasts it to a chosen group address G_{A1}. The GSI entities that are direct children of the GSC in the hierarchy then decrypt

[2] An exception is the *flat key management scheme* presented in [CWSP98], which can be defeated by a coalition of only 2 members.

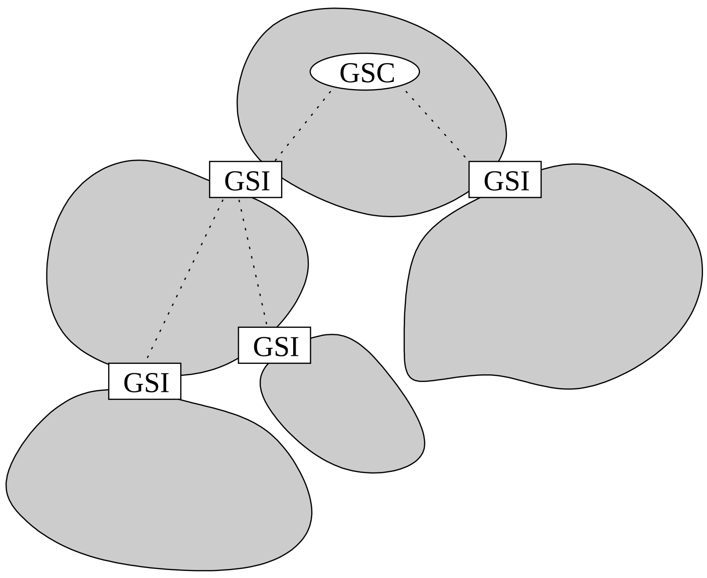

Fig. 4. IOLUS

the data and re-encrypt it with another key K_i, and broadcast it to their own multicast group G_{Ai}. This mechanism is applied recursively all the way down the tree hierarchy. Consequently, this construction can be viewed as a tree network where each node decrypts data received from its parent and re-encrypts it for its children. In this chapter we refer to this type of approach as "re-encryption trees".

Each node or GSI only needs to know the key from its parent in the hierarchy and its own subgroup key. Members in a subgroup are managed by a single GSI and only know the GSI's key K_i.

Removing a member. When a member is removed from a subgroup, the GSI chooses a new subgroup key K_i' and uses several authenticated secure unicast channels to send the new key to the remaining members in the subgroup (alternatively one long multicast message corresponding to the concatenation of these unicast messages).

Adding a member to the group. When a member is added to a subgroup we can proceed in a similar way as we did to remove a member, except that we also send the new subgroup key K_i' to the added member. An improvement in terms of overhead is to encrypt the new subgroup key K_i' with the old one K_i and multicast the message $E_{K_i}(K_i')$ to the subgroup while transmitting K_i' to the added member on a separate authenticated secure unicast channel. However, this improvement is less secure because it creates a potential chain between subsequent keys, thus the exposure of a single key has a potentially stronger impact.

10.2 Key Distribution

The transformations in an hierarchical tree we describe above operate directly on the encryption of the multicast content. As an alternative, S. Mittra also suggests to use the IOLUS framework to distribute a global short term key. This key is used to encrypt content to be transmitted on a separate autonomous multicast channel, which can be based on a different multicast routing scheme.

Hardjono et al. proposed a re-encryption tree for key distribution in IGKMP[HCD00], the Intra-Domain Group Key Management Protocol. This protocol essentially describes a 2 level re-encryption tree that is used to distribute a common group key $\mathcal{K}$ to the group.

10.3 Analysis

This scheme provides Data Confidentiality (req. 1) as well as backward and forward secrecy (req. 2). Since keys in each subgroup are chosen independently, colluding members do not seem to gain additional privileges beyond the sum of their own access, thus this scheme is collusion resistant (req. 3).

In his work on IOLUS, S. Mittra did not specify a limit to the size of a subgroup. In practice, however, we need to assume that a subgroup contains no more than M members, otherwise processing scalability issues will arise in subgroups. The benefit of IOLUS is precisely to divide a large multicast group into subgroups that are small enough so that they do not exhibit the scalability issues of a large multicast group. If subgroups are small enough then this scheme will provide processing scalability, since computations and communication overhead will depend on M and not on the full group size.

A second benefit of subgrouping is that it provides stronger membership robustness (req. 7). When a member is added or removed from the group it only affects the other members which depend from the same GSI. Members in other subgroup are not even aware that a member was added or removed from the group. Consequently, the impact of a membership change is restricted to a subgroup of at most M members.

The third benefit of subgrouping is containment (req. 4). Since only at most M members of the same multicast group use the same key K_A to access the multicast data then the exposure of K_A will only have a local impact, limited to the subgroup G_A that uses the same key K_A. If the attacker is in another subgroup G_B, out of scope of G_A than the exposed key K_A will be useless unless he also manages to forward traffic from G_A to G_B.

The main drawback of this scheme is that it fully trusts all the intermediary elements to access the multicast content, which is contrary to requirement 5. Good processing scalability, membership robustness and containment all depend on a reasonably small value of M, the size of the subgroup. But if M is small than it increases the number of Groups Security Intermediaries needed. For large multicast groups, this means that each content provider will need to construct a large network of trusted intermediaries. This adds a significant cost for the content provider.

11 MARKS

The MARKS scheme was proposed by Briscoe in [Bri99], and features a subscription based approach to multicast confidentiality. Consider a content divided in n segments $\{S_1, ..., S_n\}$ each encrypted with a different key $\{k_1, ..., k_n\}$: the source simply broadcasts $E_{k_i}(S_i)$ for each $1 \leq i \leq n$. To provide access to a member R for segments S_j through $S_{(j+l)}$ the membership manager needs to provide R with the keys $\{k_j, k_{(j+1)}, ..., k_{(j+l)}\}$. Simply sending the list of keys creates an overhead which linearly increases with l, which does not provide processing scalability (req. 6). In his work, Briscoe provides a method to get around this limitation by constructing a one-way hash tree as follows.

11.1 The Hash Tree

Let n define the number of keys or segments used in the broadcast where $n = 2^d$. Let $\mathcal{L}$ and $\mathcal{R}$ define two one-way functions. For example, we can choose $\mathcal{L}$ (*resp.* $\mathcal{R}$) to be the left (*resp. right*) half of of a pseudo-random generator which doubles its input, or we can use two independently keyed hash functions [BCK96]. We initially choose a random value u_0 and we construct a balanced binary tree as follows:

– assign u_0 to the root.
– if a node is assigned a value u_i we assign to the left child of u_i the value $\mathcal{L}(u_i)$ and to the right child the value $\mathcal{R}(u_i)$.

The leafs of the tree ordered from left to right represent the keys $\{k_1, ..., k_n\}$ used to encrypt the segments $\{S_1, ..., S_n\}$.

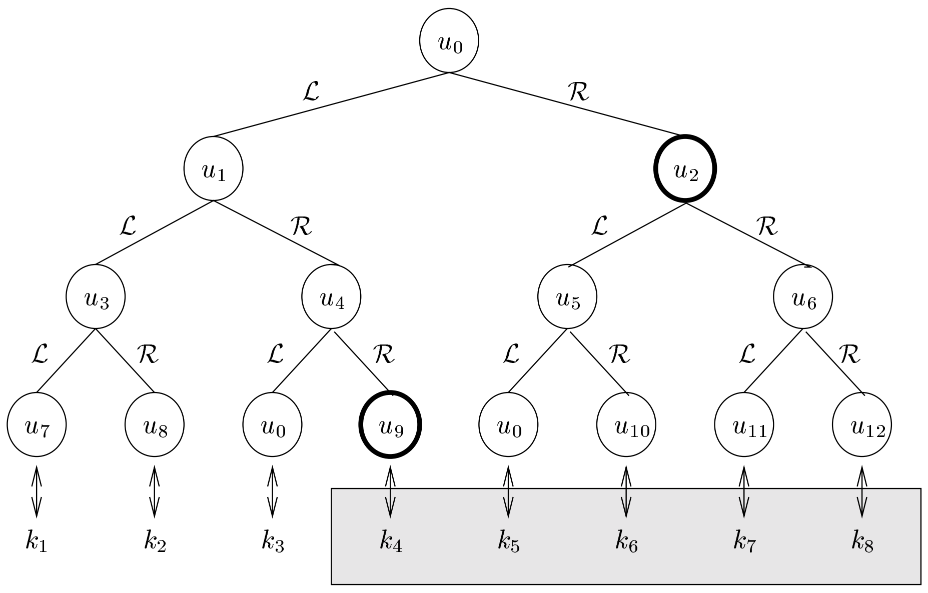

Fig. 5. MARKS with an eight segment stream.

The advantage of this construction is that any sequence $\{k_j, k_{(j+1)}..., k_{(j+l)}\}$ of l consecutive keys in $\{k_1, ..., k_n\}$ can be represented with at most $2.log_2(n) - 1$ values. Indeed, instead of describing a sequence of l keys by listing them in a sequential order, we can simply provide the points in the tree that can be used to derive the needed keys (and only those keys) by applying $\mathcal{R}$ and $\mathcal{L}$. This problem is quite similar to finding IP network aggregations in CIDR[FLYV92]. Consider the example on Figure 5 where a member subscribes for segments 4 to 8. He receives u_2 and u_{10}, next he can efficiently compute $k_4 = u_{10}$, $k_5 = u_{11} = \mathcal{L}(\mathcal{L}(u_2))$, $k_6 = \mathcal{R}(\mathcal{L}(u_2))$, $k_7 = \mathcal{L}(\mathcal{R}(u_2))$, $k_8 = \mathcal{R}(\mathcal{R}(u_2))$.

11.2 Analysis and Conclusion

In the MARKS approach, once the membership manager provides a member with access to l consecutive segments $\mathcal{S} = [S_{j+1}, ..., S_{j+l}]$, it cannot revoke this access unless it redistributes new keys to all other members whose segments intersect with $\mathcal{S}$. In practice it would require the construction of a new key tree over the segments of the stream. Revocation is not possible here without running into strong processing scalability issues. However, there are some multicast applications that will be satisfied with a non revocable subscription based approach. Consider, for example, some Pay-per-view TV applications where the content length is know in advance and the clients pay before the show is broadcasted. For such applications MARKS offers benefits such as:

– Data confidentiality, backward and forward secrecy, collusion resistance and no intermediary trust.

– Processing Scalability.
– Perfect membership robustness.

The last characteristic is the strongest point of MARKS: it does not require any message to be broadcasted to anybody when a recipient's membership expires in the group. When a member is added to the group, he only needs to receives the keys corresponding to his subscription, which can be transmitted to him in advance.

Similarly to LKH, this scheme does not provide any containment. In terms of key exposure, the LKH has a little advantage over MARKS, because the membership can invalidate an exposed key easily, while MARKS by definition does not provide any practical way to do this.

12 Multicast Confidentiality Summary

The three types of multicast confidentiality schemes reviewed in this chapter all suffer from some limitations. The LKH scheme does not offer containment and has some membership robustness issues though some progress has recently been made in that area. The re-encryption tree approach relies on a potentially large infrastructure of fully trusted intermediary elements. Finally the MARKS scheme does not allow member revocation prior to the end of their subscription, neither does it offer any form of containment.

13 Conclusion

Security in multicast raises genuinely new authentication and confidentiality requirements that are not met by classical network security mechanisms. At the core of multicast confidentiality, membership management is a complex issue involving membership managers which communicate on a bidirectional channel with members in different situations. Specific membership management issues are rarely mentioned in multicast confidentiality frameworks, however we believe that these issues would deserve to be explored in future research. For example, in large commercial applications, distributed membership management will become a key requirement. One reason is scalability. Indeed, in a very large multicast group, a single membership manager serving all requests, would quickly become a bottleneck. The second reason is related to laws and regulations. Different countries have different commercial laws and sometimes different cryptography regulations. It makes sense to distribute several membership managers according to these constrains. Re-encryption trees are well suited for distributed membership management, and the MARKS subscription approach is even better. However, it seems that a construction such as LKH is inherently hard to distribute. Besides, multicast confidentiality as described in this chapter is faced with a dilemma. Approaches based on a shared key a common key among members do not offer any form of containment and piracy will greatly hinder the use of such schemes in large groups. On the other hand, the schemes that provide containment require the existence of active intermediate elements in the network with a significant a deployment cost. It is thus not clear yet what type of solution will emerge for confidentiality in large scale multicast applications. Some solutions will possibly rely on a combination of re-encryption trees for inter-domain communications in with an LKH approach for intra-domain communications. Moreover, content providers recently have shown a strong concern in protecting the content they distribute beyond the notion confidentiality we described in this chapter. As in the example of the highly publicized lawsuit[USDC99] that opposed the RIAA to Napster, the mp3 audio download service, issues such as copy protection and copyright have become paramount for content distributors. Architectures that provide these additional services typically rely on several cooperating tamper-proof hardware components achieving end to end protection of the content between the source and the ultimate video or audio playback hardware (see for example, CPRM, `http://www.4Centity.com/tech/cprm/`). Future commercial multicast confidentiality frameworks will potentially rely on such tamper-proof hardware platforms.

References

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *Lecture Notes in CS*, 1109:1–15, 1996.

[BD95] M. V. D. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In Alfredo De Santis, editor, *Advances in Cryptology - EuroCrypt '94*, pages 275–286, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science Volume 950.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *IEEE Symposium on Foundations of Computer Science*, pages 394–403, 1997.

[Bri99] Bob Briscoe. MARKS: Zero side-effect multicast key management using arbitrarily revealed key sequences. In *First International Workshop on Networked Group Communication*, November 1999.

[CEK$^+$99] Isabella Chang, Robert Engel, Dilip Kandlur, Dimitrios Pendarakis, and Debanjan Saha. Key management for secure internet multicast using boolean function minimization techniques. In *Proceedings IEEE Infocomm'99*, volume 2, pages 689–698, March 1999.

[CGI$^+$99] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of IEEE Infocom'99*, 1999.

[CWSP98] Germano Caronni, Marcel Waldvogel, Dan Sun, and Berhardt Plattner. Efficient security for large and dynamic multicast groups. In *IEEE 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '98)*, 1998.

[Dee89] Steve E. Deering. RFC 1112: Host extensions for IP multicasting, Aug 1989.

[Dee91] Steve E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[Fen97] W. Fenner. Internet group management protocol, version 2. Request For Comments 2236, November 1997. see also draft-ietf-idmr-igmpv3-and-routing-01.txt for IGMP v3.

[FLYV92] V. Fuller, T. Li, J. Yu, and K. Varadhan. Supernetting: an address assignment and aggregation strategy, 1992.

[FN93] A. Fiat and M. Naor. Broadcast encryption. In Douglas R. Stinson, editor, *Advances in Cryptology - Crypto '93*, pages 480–491, Berlin, 1993. Springer-Verlag. Lecture Notes in Computer Science Volume 773.

[HCD00] Thomas Hardjono, Brad Cain, and Naganand Doraswamy. A framework for group key management for multicast security. Internet-Draft, work in progress, February 2000.

[LYGL01] Xiaozhou Steve Li, Yang Richard Yang, Mohamed G. Gouda, and Simon S. Lam. Batch rekeying for secure group communications. In *Tenth International World Wide Web conference*, pages 525–534, 2001.

[Mit97] Suivo Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM'97 (September 14-18, 1997, Cannes, France)*, 1997.

[Moy94] John Moy. Multicast extensions to OSPF. Request For Comment 1584, March 1994.

[MS97] T. Maufer and C. Semeria. Introduction to IP multicast routing. Internet-Draft, July 1997. draft-ietf-mboned-intro-multicast-03.txt.

[MS98] David A. McGrew and Alan T. Sherman. Key establishment in large dynamic groups using one-way function trees. Technical report, TIS Labs at Network Associates, Inc., Glenwood, MD, 1998.

[oST01] National Institute of Standards and Technology. Advanced Encryption Standard, 2001.

[PB99] R. Poovendran and John S. Baras. An information theoretic analysis of rooted-tree based secure multicast key distribution schemes. In *CRYPTO*, pages 624–638, 1999.

[PST01] Adrian Perrig, Dawn Song, and Doug Tygar. ELK, a new protocol for efficient large-group key distribution. In *IEEE Symposium on Security and Privacy*, May 2001.

[Riv95] R.L. Rivest. The RC5 encryption algorithm. In In B. Preneel, editor, *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96. Springer Verlag, 1995.

[STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM Conference on Communications Security (March 14-16, 1996, New Delhi, India)*, 1996.

[STW98] Michael Steiner, Gene Tsudik, and Michael Waidner. CLIQUES: A new approach to group key agreement. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, pages 380–387, Amsterdam, May 1998. IEEECSP.

[USDC99] Northern District of California United States District Court. RIAA v. Napster, court's 512(a) ruling, 1999.

[WGL98] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. In *ACM SIGCOMM 1998*, pages 68–79, 1998.

[WHA98] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key management for multicast: Issues and architectures. Internet draft, Network working group, september 1998, 1998.

[WL00] C. Wong and S. Lam. Keystone: a group key management system. In *Proceedings of International Conference in Telecommunications*, 2000.