A Secure Multicast Protocol with Copyright Protection

Hao-hua Chu, Lintian Qiao and Klara Nahrstedt

Department of Computer Science, University of Illinois at Urbana-Champaign,

1304 West Springfield Avenue, Urbana, IL 61801, U.S.A.

ABSTRACT

We present a simple, efficient, and secure multicast protocol with copyright protection in an open and insecure network environment. There is a wide variety of multimedia applications that can benefit from using our secure multicast protocol, e.g., the commercial pay-per-view video multicast, or highly secure military intelligence video conference. Our secure multicast protocol is designed to achieve the following goals. (1) It can run in any open network environment. It does not rely on any security mechanism on intermediate network switches or routers. (2) It can be built on top of any existing multicast architecture. (3) Our key distribution protocol is both secure and robust in the presence of long delay or membership message. (4) It can support dynamic group membership, e.g., JOIN/LEAVE/EXPEL operations, in a network bandwidth efficient manner. (5) It can provide copyright protection for the information provider. (6) It can help to identify insiders in the multicast session who are leaking information to the outside world.

Keywords: Multicast security, copyright protection, key distribution

1. INTRODUCTION

We present a simple, efficient, and secure multicast protocol with copyright protection in an open and insecure network environment. There is a wide range of multimedia applications that can benefit from using our secure multicast protocol, e.g., the commercial pay-per-view video multicast, or highly secure military intelligence video conference. Our secure multicast protocol is designed to achieve the following goals:

- *Security in Open Network Environment*
  We assume that group members, who can be *either or both senders and receivers*, are in an open network environment. This means that the multicast streams may travel through intermediate switches or routers which may or may not have any security mechanism. Therefore, our secure multicast protocol must not depend on any of the intermediate network components for security support. For example, the Scalable Multicast Key Distribution by Ballardie[1] does not satisfy this property. It is based on the Core Based Tree architecture[2] which requires the network routers to maintain a hard state, including the security information, on the nodes of the multicast delivery tree.

- *Multicast Architecture Independence*
  Our secure multicast protocol can be implemented on top of any existing multicast protocols: M-OSPF,[3] DVMRP,[3] CBT,[2] or PIM.[4] We achieve this by encrypting or decrypting data on the *endpoint hosts* before sending it to or after receiving it from the underlying multicast protocol.

- *Robust Dynamic Membership Support*
  Lost packets and long network delay are prevalent in any open network environment, e.g. the Internet, where the traffic congestion level and bandwidth availability for members in the same multicast group can vary significantly. As a result, the key distribution protocol must deal gracefully with lossy or long delay unreliable multicast channels. For example, the group key management protocol by Wallner[5] uses the same multicast channel to distribute the group key as well as the multicast data. It is very likely that some members in the multicast group do not get the key update (rekey) messages in time because of lost messages or long network delay. This can create windows of security vulnerability. For example, a recently expelled receivers may still be able to decrypt the multicast data from senders who do not receive the new key update messages in time.

This research is supported by National Science Foundation Career Grant NSF-CCR-96-23867 and Research Board of University of Illinois at Urbana-Champaign. For further author information, e-mail h-chu3,l-qiao,klara@cs.uiuc.edu

- *Copyright Protection*
  We assume that the content provider needs to have copyright protection for multicast video data, so that the rightful ownership of the video data can be identified. We apply the watermark technique to encode the ownership information into the video data.

- *Leakers Identification*
  It is possible that some legal group members in the multicast session may leak the multicast data to non-members for free or for a profit. The leaking of this multicast stream may cause a security or copyright violation, and the consequence can be severe depending on the type of multicast applications. In case of a military intelligence conference, a spy may gain clearance to be a legal group member and then leaks the multicast content to hostile foreign agencies. By embedding an unique watermarking sequence inside the multicast stream for different receivers, our multicast protocol enables the content providers and the group leader to identify the leaker(s) after the leaking data is discovered and analyzed.

We organize the remainder of the paper as follows: section 2 describes the related work; section 3 presents our key distribution protocol; section 4 presents our multicast watermark protocol; section 5 states our conclusion; and appendix A provides a list of definitions for the various notations used in this paper.

2. RELATED WORK

2.1. Multicasting Schemes and Security Issues

The existing multicast security protocols are all focused on the problem of key management. The goal of the key management is to distribute the *group key* securely to the group members who can then use it to encrypt or decrypt the multicast data. They deal with issues like bandwidth scalability and the number of key messages exchanged with increasing group size. We will describe three different key distribution approaches.

2.1.1. Core Based Tree Key Distribution

The Core Based Tree (CBT) key distribution by Ballardie[1] is based on the *hard state* multicast protocol like the Core Based Tree[2] where the multicast routers permanently maintain the state of the multicast tree, e.g. their adjacent routers in the tree. The key distribution algorithm can take advantage of the hard state approach by appending various security information into the hard state of the tree, e.g., the access control list (ACL), the group key, and the key encrypting key (which is used for re-keying the group key). The algorithm contains the following steps: (1) the initiating host first communicates, via asymmetric encryption, the ACL to a core router, (2) the core router generates the group key and the key encrypting key, (3) when a new non-core router joins to become a part of the multicast tree, the core router authenticates the new non-core router and passes the security information using asymmetric encryption to the non-core router, and (4) as the multicast tree expands, the authenticated non-core router further authenticates other new incoming non-core routers and distributes the security information.

This distributed key distribution approach has an efficiency improvement over a centralized key distribution approach where the group key is distributed to the group members by only one or a few centralized servers. However, the security level of the CBT key distribution scheme is based on a strong assumption that the involved multicast routers can be trusted not to leak the security information. In addition, the key distribution algorithm does not address dynamic membership operations such as *JOIN/LEAVE/EXPEL*.

2.1.2. Hierarchical Tree Key Distribution

The Hierarchical tree key distribution by Wallner[5] is an efficient and scalable approach that supports dynamic group membership. The algorithm contains the following steps. (1) Each multicast group contains a key server which maintains a rooted tree structure, and each leaf node corresponds to a group member as shown in Figure 1. (2) Each node in the tree contains a key–each leaf node holds a pairwise key established between the server and the member (e.g., $K_1, K_2, .., K_8$), each intermediate node holds a key generated by the server (e.g., $K_a, K_b, ...K_f$), and the root node holds the group key which is used to encrypt the data (e.g., K_g). (3) The server sends to each group member a sequence of keys on the path from his/her leaf node to the root. (4) Each key is encrypted with the previous key in the sequence to ensure security.

We will illustrate it with the example in Figure 1. Member 1 first establishes the pairwise key K_1 with the server, and it receives the sequence of intermediate and group keys (K_a, K_e, K_g), where K_a is encrypted with K_1

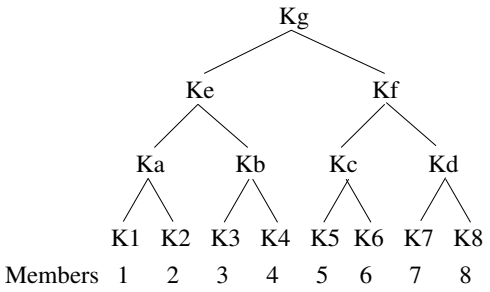

Figure 1. Hierarchical tree key distribution

($\{K_a\}_{K_1}$*), $\{K_e\}_{K_a}$, and $\{K_g\}_{K_e}$. To expel a member from the group, intermediate and group keys on the path from the expelled member's leaf node to root must be changed. For example, the removal of member 1 from the multicast group requires that the server generates new keys (K'_a, K'_e, K'_g). Each new key is encrypted using its two immediate child keys: $(\{K'_a\}_{K_2}, \{K'_e\}_{K'_a}, \{K'_e\}_{K_b}, \{K'_g\}_{K'_e}, \{K'_g\}_{K_f})$. This new sequence is assembled into one rekey message which is then multicasted to all members using a multicast channel. Upon receiving the rekey message, the members decrypt only those keys that they need and no more. For example, member 2 can decrypt (K'_a, K'_e, K'_g), members 3-4 can decrypt only (K'_e, K'_g), and members 5-8 can decrypt only (K'_g). Given a group size of N, each membership update operation *(JOIN/LEAVE/EXPEL)* requires one rekey message that contains $Log(N)$ number of keys.

However, the multicast channel used by the key distribution may be unreliable when it can suffer long network delay or even lost messages. This creates the following problems.

- *Security Loophole*
  The server multicasts a rekey message, but some senders in the group do not receive it in time. As a result, the senders continue to multicast data encrypted with the old group key. Recently expelled members can use the old group key to decrypt the data. This is considered a security violation.

- *Failure*
  The key distribution algorithm can fail with lost messages. If members miss any rekey messages, they won't be able to decode any future rekey messages as well as future multicast data. There are two possible but expensive solutions. (1) Implement a reliable multicast channel with receiver acknowledgments and retransmissions, which is similar to the concept of TCP for the unicast connection. However, a reliable multicast channel is very expensive and inefficient. The reason is high probability that a small portion of receivers are on congested networks so that the sender may have to retransmit many times to get acknowledgments from all receivers. As a result, the bandwidth overhead caused by retransmission(s) and the associated delay in acknowledgments are unacceptable, especially for multimedia applications with real time constraints. (2) Communicate the lost messages through a reliable unicast channel between each lossy member and the server. However, this is also unacceptable given that the server may have to open many reliable unicast channels to handle a sizable number of lossy members.

2.1.3. Iolus

Iolus key distribution by Mittra[6] carries the CBT distribution concept further. It divides the group into regional subgroups, and each subgroup is managed by a trusted Group Security Intermediary (GSI). Each subgroup is treated almost like a separate multicast group with its own subgroup key and its own multicast channel. The GSI in each subgroup manages its subgroup key distribution and authenticates new members joining/leaving its subgroup. The advantage is that the subgroup runs independently of each other, and the GSI can perform dynamic member operations efficiently and independently without involving members of other subgroups. To bridge data across the subgroups, the GSIs use another separate multicast channel managed by the Group Security Controller (GSC). As

*We will use the common notation $\{X\}_K$ to denote that X is encrypted with K.

a result, each data transmission requires three different multicasts. The sender first multicasts data in its subgroup channel. When the sender's GSI receives the data, it multicasts the data to the other GSIs. Then the other GSIs multicast the data to their subgroup members through their subgroups' multicast channels.

Iolus, similar to the CBT approach, depends on the security level of the various GSIs residing at various locations in the network. Its overhead contains the three multicast transmissions per data transmission, the management of multiple subgroups, and their multicast channels.

2.2. Watermarking Issues

During the past few years, a number of digital watermarking methods have been proposed. Among the earliest works, L.F. Turner[7] has proposed a digital audio watermarking method which substitutes the least significant bits of randomly selected audio samples with the bits of an identification string (watermark). Similar idea can also be applied to images.[8] There are many other proposals for watermarking such as Tanaka's schemes[9] which use the fact that the quantization noise is typically imperceptible to users, Brassil's methods[10] for textual document images, and Caronni's geometric patterns (also called tags).[11]

Craver, Memon, Yeo, and Yeung[12] address an important issue of rightful ownership. They provide an attack (*CMYY attack*) counterfeit watermarking schemes that can be performed on a watermarked image to allow multiple claims of rightful ownerships. They also define so called Non-invertible watermarking scheme. Qiao and Nahrstedt[13] address and prove the non-invertibility property. Schemes applying to MPEG video stream and uncompressed video stream are designed.

3. KEY DISTRIBUTION ALGORITHM

Our key distribution algorithm is designed to achieve the goals described in section 1: (1) security in open network environment, (2) multicast architecture independence, and (3) robust dynamic membership support.

To startup a new secure multicast group, our key distribution algorithm requires only a *group leader* to be started. The *group leader* has the authority and the necessary information to accept/reject new membership requests. For example, the group leader may be given an access control list (ACL) which it can check if the new member can join it, or it can accept and verify a payment information as a mean for new members to be admitted into the multicast group. We also assume that the address of the *group leader* is known to anyone who is interested to join the secure multicast group.

To join a secure multicast group, a potential member first sends a *JOIN* request to the group leader using a secure unicast channel. The group leader checks, e.g. its ACL, to decide to either accept or reject the *JOIN* request. If the *JOIN* request is accepted, the group leader generates a unique member id *uid* that identifies the new member. The member id is then communicated to the new member who can then begin to send and to receive data according to the steps described in the following subsection.

3.1. Data Transmission

Data transmission can be divided into three phases as shown in Figure 2: (1) the *sending phase* when the sender multicasts an encrypted data message, (2) the *verification phase* when the group leader multicasts a verification message that contains the key for decrypting the data message, and (3) the *receiving phase* when the receivers receive both the data and verification messages and decrypt the data. We now describe these three phases in details.

3.1.1. The Sending Phase

The first stage involves the sender constructing a data message that contains 3 components as shown in Figure 2 (step S1). The first component contains the sender's member id ($suid$) and a message id ($msgid$). Each sender maintains a $msgid$ counter, which is initialized to 0 and is incremented for every new message created. The $(suid, msgid)$ pairs uniquely identify a message in the multicast session.

The second component of the message contains the data encrypted (via symmetric encryption) with a message key K_{msg}. The key is used only once for the current message, and a new key is randomly generated by the sender for the next message. The key generation can use any secure key generation algorithms which satisfy the property that new keys cannot be predicted from the previous and subsequent generated keys.

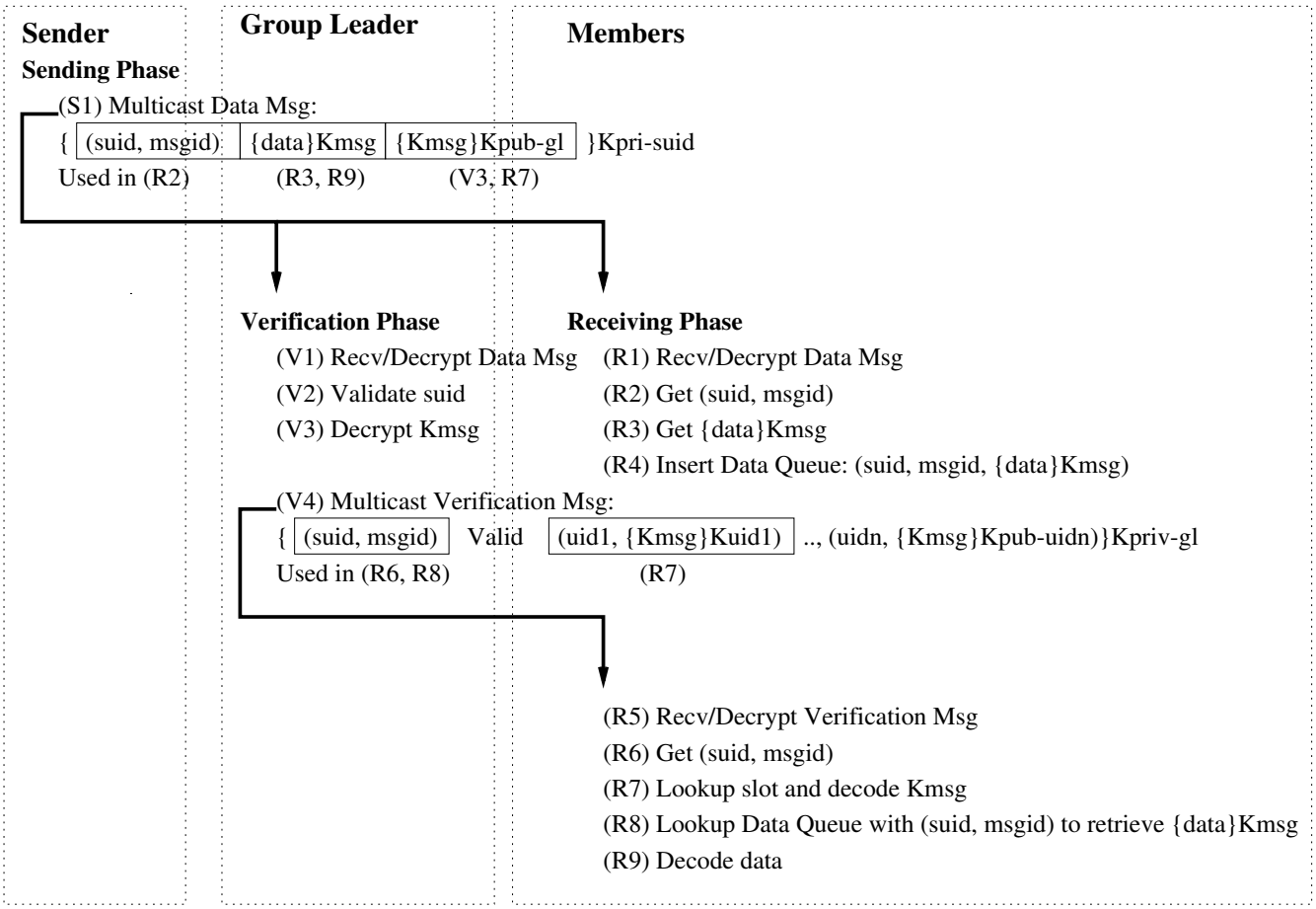

Figure 2. Three phases of our secure data transmission.

The third component of the message contains the message key K_{msg} encrypted with the public key of the group leader (K_{gl}^{pub}). All three components are assembled into one *data message* which is then signed with the private key of the sender (K_{suid}^{pri}). The sender multicasts the following data message in the multicast channel:

$$\{(suid, msgid), \{data\}_{K_{msg}}, \{K_{msg}\}_{K_{gl}^{pub}}\}_{K_{suid}^{pri}}$$

When the data message is received from the multicast channel, members cannot decode the data yet because they do not have the message key K_{msg}. Only the group leader has K_{gl}^{pri} which can be used to decode K_{msg}.

3.1.2. The Verification Phase

The second phase involves the group leader as shown in Figure 2 (steps V1-V4). Upon receiving the data message from the sender, the group leader decrypts the data message with the known public key of the sender in (V1). Then it looks up its current membership list to check that the sender is indeed a valid group member in (V2). It also decrypts the third component of the data message using its private key to reveal the message key K_{msg} in (V3).

When the validation check succeeds, the group leader prepares a verification message which contains three components. The first component is the message tag $(suid, msgid)$. The second component is a $VALID$ symbol indicating that the data message $(suid, msgid)$ has been verified by the group leader. The third component contains a sequence of *slots* where each valid member in the current membership list has a corresponding slot. The slot contains the

pairs uid and K_{msg} encrypted with the public key of the corresponding member K_{uid}^{pub}, so that only member uid can decrypt K_{msg} from reading this slot.

All three components are assembled into one *verification message* which is then signed with the private key of the group leader (K_{gl}^{pri}). The group leader multicasts the following valid verification message in the multicast channel in (V4).

$$\{(suid, msgid), VALID, \quad (uid_1, \{K_{msg}\}_{K_{uid_1}^{pub}}), \\ (uid_2, \{K_{msg}\}_{K_{uid_2}^{pub}}), .., \\ (uid_n, \{K_{msg}\}_{K_{uid_n}^{pub}})\}_{K_{gl}^{pri}}$$

It is also possible that one of the verification checks fails, e.g., the sender does not belong the group. The group leader prepares an invalid verification message containing the message tag $(suid, msgid)$ and an $INVALID$ symbol. The verification message is signed with the private key of the group leader. Since the data message is invalid, there is no need for the receivers to decrypt the counterpart data message. Hence the message key K_{msg} is not included in the verification message. The group leader multicasts the following invalid verification message:

$$\{(suid, msgid), INVALID\}_{K_{gl}^{pri}}$$

3.1.3. The Receiving Phase

The third phase involves the receiver listening on the multicast channel as shown in Figure 2 (steps R1-R9). Upon receiving a data message, the receiver decrypts it with the public key of the sender to reveal the message tag $(suid, msgid)$ in (R1,R2). Since the receiver may not have received its counterpart verification message which contains the necessary message key K_{msg} to decode the data, the receiver stores the encrypted *data component* $\{data\}_{K_{msg}}$ along with the message tag as its lookup index in a *data queue* in (R3-R4).

Upon receiving a verification message, the receiver decrypts it with the public key of the group leader to reveal the message tag $(suid, msgid)$ in (R5-R6). If the verification message contains the $VALID$ symbol, the receiver uses his/her assigned uid and searches for his/her slot that contains the message key K_{msg} in the verification message in (R7). After the message key is decrypted, the receiver uses the message tag to retrieve the corresponding encrypted data component from the data queue in (R8). Then the receiver can decrypt the data with the message key in (R9). If the verification message contains the $INVALID$ symbol, the receiver simply removes the corresponding encrypted data component from the data queue.

Two other scenarios can arise due to the variable network delay or lost messages. (1) The receiver may receive a verification message before its counterpart data message arrives. Then the receiver needs to buffer the verification message in a *verification queue* till its data message arrives. (2) Some data or verification messages may get lost in the network so that some receivers may never receive them. As a result, the counterpart message to that lost message may remain in the queues forever. For example, if a data message is lost but the counterpart verification message is received, the verification message may remain in the verification queue forever. The receiver can use a *time-out-and-drop* or *overflow-and-drop* policy to maintain a bounded queue size: if a message remains in the queue for more than some time period, it is dropped; or if the message queue exceeds a size limit, the earliest arrived message is dropped.

3.2. Dynamic Membership Operations

Our secure multicast protocol supports three dynamic membership operations: a potential member can *JOIN* the group, an existing member can *LEAVE* the group, and the group leader can *EXPEL* an existing member. For the *JOIN* operation, the group leader allocates a new member id to the new member, and adds an additional message key slot $(uid, \{K_{msg}\}_{K_{uid}^{pub}})$ into the verification messages. For the *LEAVE* and *EXPEL* operations, the group leader removes the message key slot corresponding to the leaving or expelled member from the verification messages. These operations are simple without any additional rekey messages or computational overhead to the existing group members and the group leader.

Our key distribution protocol can also support a *SUSPEND* operation which denies access to a group member for a time duration. For the *SUSPEND* operation, the group leader temporarily removes the message key slots corresponding the suspended members from the verification messages. This does not require any additional *re-JOIN* and re-authentication operations. The decision of suspension can be made by the group leader based on the credentials (e.g. age) of the members. The *SUSPEND* operation is very useful for a video-on-demand multicast where minors are temporarily suspended from seeing inappropriate material.

We will show that our key distribution algorithm is robust and secure in the presence of long network delay or lost messages. Note that a lost message is equivalent to a message suffering an infinitely long delay. We consider the following two scenarios. (1) The group leader does not receive the data message in time, so it will not multicast the counterpart verification message which contains the message key to the data message. As a result, no members can decrypt the counterpart data message and it will be eventually dropped from the members' data queues according to the time-out-and-drop or the overflow-and-drop policy. Our security policy does not guarantee that all valid messages are received by the members. However, it can guarantee that expelled or non-members cannot decode any data in the presence of long network delay or lost messages. (2) Some receivers do not receive either the data or the verification message in time so that they won't be able to decrypt that lost message. However, since our key distribution algorithm uses a new key for every new data message, a lost message has no adverse effects on the future messages.

In contrast, other existing key distribution protocols, e.g. the Hierarchical Tree Key Distribution,[5] can fail in the presence of long network delay or lost messages.

1. Some senders do not get the rekey message, and they continue to encrypt data with the outdated group key. As a result, expelled receivers can still decrypt the data.
2. Some receivers do not get the rekey message. When expelled senders send data using the outdated group key, these receivers receive the data which is in fact invalid.
3. Some receivers do not get some rekey messages. Because of the algorithmic dependency among the subsequent rekey messages, they will not be able to process any future rekey messages and to decrypt any future data.

We also note that our key distribution protocol does not depend on any intermediate nodes for security. The group members authenticate directly with the group leader through secure unicast connections when they first join. The secure unicast connections can be closed after the authentication process and they are no longer used during data transmission. The encryption and decryption are done at the endpoint hosts only. Our key distribution protocol applies encryption on the data before sending it to the underlying multicast protocol, and it applies decryption on the data after receiving it from the underlying multicast protocol. This means that our key distribution protocol can be implemented on top of any multicast architecture. Our end-host solution is especially applicable to the M-OSPF or DVMRP multicast architecture where the multicast server simply floods the multicast messages across network, and the multicast messages are available to everyone listening over the network.

3.2.1. Overhead Analysis

The dominating network bandwidth overhead in our key distribution algorithm is in the verification messages. The size of the verification message grows linearly with the group size N because it contains N copies of message keys, each is encrypted for each group member. The verification message is multicasted once for every data message multicasted. Let M be the number of data messages multicasted per second (or the $message\_rate$), the network bandwidth overhead is $O(N * M)$.

The storage requirement at the group member consists of (1) the public keys of all the senders and the group leader, (2) his/her assigned member id, and (3) two queues for the data/verification messages. The number of senders is usually very small (a constant) relative to the size of the group, e.g. the pay-per-view video application has only one sender. The size of message queue is bounded by a constant due to the time-out-and-drop or the overflow-and-drop policy described in section 3.1.3.

At a first glance, our key distribution algorithm does not seem to be scalable in terms of network bandwidth overhead in comparison with other optimal secure multicast protocols. But this turns out to be false. We compare our

Table 1. Overhead comparison between our key distribution protocol and the Hierarchical Tree protocol.

	Our Protocol	Hierarchical Tree Protocol
Network bandwidth overhead	$O(N * M)$	$O(p * N * log(N))$
Storage overhead	$O(1)$	$O(log(N))$

overhead with the Hierarchical Tree Key Distribution,[5] which is considered one of the most optimal key distribution algorithm in Table 1.

In the Hierarchical Tree protocol, the dominating overhead is in the rekey message which has a size $O(log(N))$. The rekey message is multicasted for every membership change. Let p be a percentage of members who are leaving or joining the group per second, the number of membership change per second in a group of size N is $p * N$. This results in $p * N$ rekey messages generated per second, and the network bandwidth overhead is $O(p * N * log(N))$. As for the storage overhead, each member needs to store the keys from its leaf to the root which is $O(log(N))$.

For the network bandwidth overhead comparison between our protocol and the Hierarchical Tree protocol, it is $O(N * M)$ vs. $O(p * N * log(N))$. Factoring out N and the constant p, it becomes M vs. $log(N)$. M is the message rate which is usually a constant. For example, a standard pay-per-view MPEG-2 video multicast runs at 30 frames or second or its message rate. Because our key distribution algorithm uses one key message per data message, the number of key messages is equal to the number of data messages. Given a reasonable large group size, $M = 30$ is as good as $log(N)$. The storage overhead is small in both protocols.

We calculate the approximate network bandwidth overhead for the group size ranging from 100 to 10000 for a MPEG-2 video multicast session in the unoptimized column of Table 2. Given that the length of a standard secure key is 128 bits and additional 2 bytes (16 bits) are used to encode the member id, the size of the verification message is $144 * N$ bits. The pay-per-view MPEG-2 video multicast has a bandwidth requirement of 4 Mbps, and it plays at 30 frames per second. Note that the overhead ratio is computed as the network bandwidth overhead over the MPEG-2 bitrate.

Table 2. Network bandwidth overhead of our key distribution algorithm given a pay-per-view MPEG-2 video multicast application and a group size ranging from 100 to 10000. The MPEG-2 video has a bit-rate of 4Mbps with a framerate of 30. We also assume that MPEG-2 stream has the repeating IBBPBBPBB pattern which one I frame occurs in every 9 frames. The optimized columns compute the overhead based on one message key per I frame, whereas the unoptimized column compute the overhead based on one message key per frame.

	Unoptimized			Optimized		
Group size	100	1,000	10,000	100	1,000	10,000
Network bandwidth overhead	432Kbps	4.32Mbps	43.2Mbps	48Kbps	480Kbps	4.8Mbps
Overhead ratio	10.8%	108%	1,080%	1.2%	12%	120%

3.2.2. Optimization

We can make a tradeoff between the network bandwidth overhead and the security level by reusing the same message key for data messages within some fixed time period of t second(s). This means that $(message\_rate * t = m)$ number of data messages share the same message key for data encryption and decryption. Therefore we multicast only one verification message every m data messages. This translates into m folds reduction in network bandwidth overhead with a tradeoff of t second(s) in *security vulnerability*. We define security vulnerability as that a recently expelled member may still be able to decrypt at most t seconds of video after the time he/she is expelled by using his/her last message key. This also holds true for a recently joined member who can decrypt at most t seconds of past video before the time he/she joins by using his/her first message key. These few seconds of security vulnerability are acceptable to many applications that do not have stringent security requirements.

The sender can also take advantage of the MPEG encoding dependency among the IPB frames to encrypt only the I frames. Without the I frames, the receivers can hardly decode the PB frames. As a result, our group leader

only needs to multicast a verification message per I-frame data message. Given a typical MPEG frame pattern of IBBPBBPBB, an I frame occurs only once every 9 frames. This translates into 9 folds reduction in network bandwidth overhead as shown in the optimized column of Table 2.

4. MULTICAST WATERMARK PROTOCOL

The copyright protection problem in the multicast environment raises an interesting issue not found in the unicast environment. In the multicast environment, all group members receive the same multicasted watermark data. When some security-sensitive data is illegally leaked out to the public, which receiver(s) are to be blamed? This is called the *leaker(s) identification* problem. In the unicast environment, this problem can be solved by the content provider sending a different watermarked copy to each different receiver. When the content provider discovers the leaked copy in the public, he/she can analyze its watermark to identify the source of the leaked copy: the receiver who is sent that particular watermarked copy by the the content provider. However, in the multicast environment, there is no way to differentiate among the receivers because they are given the same multicast copy. As a result, there is no way to identify the leaker(s).

Leaker(s) identification can be a very powerful tool to protect copyright in a secure multicast environment. Take the example of the pay-per-view video multicast. A leaker can join the multicast session as a legal customer. Once the leaker receives the data, he/she re-distributes or resells the data to the public. Our multicast watermark protocol enables the content provider to identify the leaker(s) in the multicast group when the leaked copy is discovered. This is a *preventive* method. Knowing that they may be caught, potential leakers may think twice before leaking out the data. Another example is a top-secret video/audio conferencing among military intelligence agents. Some agents may be spies who are selling the video/audio content to hostile foreign agencies. The leaker(s) identification can help to catch the spies.

Our multicast watermark protocol is a *direct extension* of our key distribution protocol described in section 3. We present the multicast watermark protocol in a similar fashion as the key distribution protocol. We first describe the extension to the *data transmission*, followed by the *leakers identification* algorithm, and then the *overhead analysis* of the multicast watermark protocol.

4.1. Data Transmission

The data transmission can be divided into five steps which are described below. Given that it is a direct extension of our key distribution protocol, we simply embed the functions of our multicast watermark protocol inside the data transmission of our key distribution protocol.

1. The sender multicasts the stream of video frames denoted as $d_1, d_2, ..., d_n$. It applies two different watermark functions to generate two different watermarked frames, d_i^{w0} and d_i^{w1}, for every picture frame d_i in the stream. The watermark generation function can be applied to the video stream prior to any video transmissions as in the case of a pay-per-view video multicast when the video stream is available. Or it can be applied just prior to each picture transmission as in the case of a live video conferencing.

2. The group leader generates a *random bit string*, denoted B_{uid}, for each member (uid) in the group.

$$B_{uid} = b_{uid}^1, b_{uid}^2, b_{uid}^3, ...b_{uid}^n$$

   The length of the bit string (denoted n) is equal to the number of video frames in the stream. In case of a live video, we use a function to generate the bit string on the fly. Each bit b_i has a value of either 0 or 1 which means that the member will be able to decrypt either the first or second (d_i^{w0} or d_i^{w1}) watermarked frame.

3. During the sending phase, the sender prepares the data message that contains two different watermarked frames, d_i^{w0} and d_i^{w1}. The format of the augmented data message[†] also contains the two corresponding message keys, K_{msg}^{w0} and K_{msg}^{w1}, which are used to decrypt the two watermarked frames.

$$\{(suid, msgid), \{d_i^{w0}\}_{K_{msg}^{w0}}, \{d_i^{w1}\}_{K_{msg}^{w1}}, \{K_{msg}^{w0}, K_{msg}^{w1}\}_{K_{gl}^{pub}}\}_{K_{suid}^{pri}}$$

[†]The original data message for our key distribution algorithm is described in section 3.1.1.

4. During the verification phase, the group leader prepares the following augmented verification message [‡] based on the random bit strings of group members. If the bit $b_i(uid)$ in the random bit string is 0 for the group member uid, then the message key slot corresponding to member uid in the verification message contains the bit value 0 and the message key to the first watermarked frame. Member uid will only be able to decrypt the first watermarked frame d_i^{w0} and not the second watermarked frame d_i^{w1}.

$$\begin{aligned}\{(suid, msgid), VALID, \quad & (uid_1, \{b^i_{uid_1}, K^{b^i_{uid_1}}_{msg}\}_{K^{pub}_{uid_1}}),\\ & (uid_2, \{b^i_{uid_2}, K^{b^i_{uid_2}}_{msg}\}_{K^{pub}_{uid_2}}), .., \\ & (uid_n, \{b^i_{uid_n}, K^{b^i_{uid_n}}_{msg}\}_{K^{pub}_{uid_n}})\}_{K^{pri}_{gl}}\end{aligned}$$

We will illustrate with an example of a group size of 4 with their member ids $uid_1, ..uid_4$. The group leader generates the following random bit strings for the group members:

frame number	1	2	3	4	5
B_{uid_1}	1	1	1	0	0
B_{uid_2}	1	0	0	1	1
B_{uid_3}	0	1	0	1	0
B_{uid_4}	0	0	1	0	1

The verification message corresponding to the 2nd data frame is as follows.

$$\begin{aligned}\{(suid, msgid), VALID, \quad & (uid_1, \{1, K^{w1}_{msg}\}_{K^{pub}_{uid_1}}),\\ & (uid_2, \{0, K^{w0}_{msg}\}_{K^{pub}_{uid_2}}),\\ & (uid_3, \{1, K^{w1}_{msg}\}_{K^{pub}_{uid_3}}),\\ & (uid_4, \{0, K^{w0}_{msg}\}_{K^{pub}_{uid_4}})\}_{K^{pri}_{gl}}\end{aligned}$$

5. During the receiving phase, the receiver decrypts its slot in the verification message to reveal the bit value and the corresponding watermark message key. Then the receiver can decrypt the data message to reveal either one of the watermarked data frame.

4.2. Leakers Identification

The leakers identification algorithm requires an input of a partial or complete leaked watermark data stream. It also requires the cooperation between the senders, who can read the watermark to produce the embedded bit string of the leaked data stream, and the group leader, who has the randomly generated bit strings of all the group members. We first assume that there exists only one leaking member. However, there is a possibility of a *collusion* if more than one members cooperate together to generate a leaked stream using a combination of their watermark streams. We first describe the algorithm for the simple case without collusion, then we describe an improved algorithm for detecting collusions.

1. The leaked watermark data stream is given to the sender(s) who analyze the watermark in the leaked stream to produce its bit string (B_{leaked}). For example, if the first frame in the illegal stream is encoded with the first watermark, then the first bit is marked with 0 ($b^1_{leaked} = 0$). If some frames are missing in the leaked stream, that bit is noted as missing '−'.

2. B_{leaked} is communicated to the group leader. The group leader performs matching between B_{leaked} and the random bit streams of the group members B_{uid}. Assuming no collusions, if one member's B_{uid} exactly matches the B_{leaked}, he/she must be the leaker.

[‡]The original verification message for our key distribution algorithm is described in section 3.1.2.

The bit string matching algorithm requires on average $2 * N$ number of bit comparisons, where N is the size of the group. Because we use random number generator to generate the bit strings, on average half of the B_{uid}s will not match B_{leaked} on a bit comparison. Assuming no collusions, members with B_{uid}s that do not match the B_{leaked} cannot possibly generate the leaked stream; therefore we can remove these B_{uid}s from the list of possible candidates for the leaker. The number of bit comparisons is calculated as follows:

$$N + N/2 + N/4... + 1 = 2 * N$$

We illustrate the matching algorithm with the following example. It has the bit strings of a leaked stream and 4 group members ($N = 4$). After the first bit comparison, we can remove uid_3 and uid_4 from the list of candidates because their first bit values do not match that of B_{leaked}. After the second bit comparison, we can further remove uid_2 from the list of candidates because its second bit value does not match that of B_{leaked}. This leaves only uid_1 who is identified as the leaker.

frame number	1	2	3	4	5
B_{leaked}	1	1	-	-	-
B_{uid_1}	1	1	1	0	0
B_{uid_2}	1	0	0	1	1
B_{uid_3}	0	1	0	1	0
B_{uid_4}	0	0	1	0	1

4.2.1. Collusion

A collusion is defined as more than one group members cooperating together to generate a leaking stream. If we consider the possibility of a collusion in the previous example, members (uid_2, uid_3), or (uid_2, uid_3, uid_4) can cooperate together to generate the first two bits in B_{leaker}.

To detect a collusion, we need to analyze more bits in B_{leaked}. Let c be maximum number of members involved in a collusion. We use the bit strings from the following example and we also assume that $c = 2$.

frame number	1	2	3	4	5
B_{leaked}	1	1	1	1	1
B_{uid_1}	1	1	1	0	0
B_{uid_2}	1	0	0	1	1
B_{uid_3}	0	1	0	1	0
B_{uid_4}	0	0	1	0	1

The leakers identification algorithm first generates a list (denoted L) consisting of all possible 2-combinations from the set of members: $L = \{(uid_1, uid_2), (uid_1, uid_3), (uid_1, uid_4), (uid_2, uid_3), (uid_2, uid_4), (uid_3, uid_4)\}$. After the first bit comparison, we can remove the combination (uid_3, uid_4) from L because uid_3 and uid_4 cannot possibly come up with the watermarked frame d_1^{w1}. Applying the same logic repeatedly, we can remove the combination (uid_2, uid_4) from L after the 2nd bit comparison, (uid_2, uid_3) after the 3rd bit comparison, (uid_1, uid_4) after the 4th bit comparison, and (uid_1, uid_3) after the 5th bit comparison. Now L has only one combination left (uid_1, uid_2). We have identified (uid_1, uid_2) as the cooperating leakers assuming $c = 2$.

We generalize the leaker(s) identification algorithm to c-collusion detection given N members. The algorithm first initializes L to contain a list of all possible c-combinations among the N members. After every bit comparison, we remove from L all the c-combinations that cannot produce the B_{leaked} bit value. Because of the complexity in the algorithm, we haven't been able to derive any formula to compute the minimum number of bits that we need to compare in order to guarantee a c-collusion detection. This is for future work. However, we can eliminate more combinations from L with a longer $|B_{leaked}|$.

4.3. Overhead Analysis

Our multicast watermark protocol puts two copies of data in the data message. Therefore, the network bandwidth overhead is approximately doubled. The storage overhead remains unchanged.

5. CONCLUSION

In this paper, we present a secure multicast protocol with copyright protection. Our secure multicast protocol contains two components: the key distribution protocol and a multicast watermark protocol. Both protocols do not require any security mechanism on the network switches or routers. They can be implemented on top of any existing multicast architecture. They are robust in the presence of long delay and lost messages when the underlying multicast protocol is unreliable. They are also efficient in terms of network bandwidth and storage requirements with dynamic membership support.

APPENDIX A. NOTATION

uid_i: the member id assigned by the group leader.
$suid$: the sender's member id.
$msgid$: the message id assigned by the sender.
N: the size of the group.
c: the number of members in a collusion.
L: a list containing the possible combinations of members in a collusion.
d_i^{w0}, d_i^{w1}: the first/second watermarked frame corresponding to the i-th data frame.
K_{msg}: key to the data message.
$K_{msg}^{w0}, K_{msg}^{w1}$: key to the first/second watermark frame in the data message.
$K_{gl}^{pub}, K_{gl}^{pri}$: public/private key of the group leader.
$K_{uid_i}^{pub}, K_{uid_i}^{pri}$: public/private key of the member uid_i.
B_{leaked}: the bit string corresponding to the leaked stream.
B_{uidi}: the bit string corresponding to the member uid.
b^i: the i-th bit value in a bit string.
b_{uid}^i: the i-th bit value in a bit string of member uid.

REFERENCES

1. A. Ballardie, "Scalable multicast key distribution," rfc1949, May 1996.
2. A. Ballardie, "Core based trees (CBT) multicast architecture," ietf draft, February 1996.
3. D. Waitzman, S. Deering, and C. Partridge, "Distance Vector Multicast Routing Protocol," rfc1075, November 1988.
4. S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, "The PIM Architecture for Wide-Area Multicast Routing," in *IEEE/ACM Transactions on Networking*, vol. 4, April 1996.
5. D. Wallner, E. Harder, and R. Agee, "Key Management for Multicast: Issues and Architectures," ietf draft, JULY 1997.
6. S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting," in *Proceedings of ACM SIGCOMM '97*, (Cannes, France), September 1997.
7. L. F. Turner, "Digital Data Security System." Patent IPN WO 89/08915, 1989.
8. R. G. van Schyndel, A. Z. Tirkel, and C. F. Osborne, "A Digital Watermark," in *Proceedings of the International Conference on Image Processing*, vol. 2, pp. 86–90, (IEEE), 1994.
9. K. Tanaka, Y. Nakamura, and K. Matsui, "Embedding Secret Information into a Dithered Multi-level Image," in *Proceedings of 1990 IEEE Military Communications Conference*, pp. 216–220, 1990.
10. J. Brassil, S. Low, N. Maxemchuk, and L. O'Gorman, "Electronic marking and identification techniques to discourage document copying," in *Proceedings of IEEE INFOCOM'94*, vol. 3, pp. 1278–1287, (Toronto), June 1994.
11. G. Caronni, "Assuring Ownership Rights for Digital Images," in *Proceedings of Reliable IT Systems, VIS'95*, Vieweg Publishing Company, 1995.
12. S. Craver, N. Memon, B. Yeo, and M. Yeung, "Can invisible watermarks resolve rightful ownerships?," in *Proceedings of the IS&T/SPIE Conference on Storage and Retrieval for Image and Video Databases V*, vol. 3022, pp. 310–321, (San Jose, CA), February 1997.
13. L. Qiao and K. Nahrstedt, "Watermarking Method for MPEG Encoded Video: Towards Resolving Rightful Ownership," in *IEEE Multimedia Computing and Systems*, (Austin, Texas), June 1998.