A Secure Multicast Protocol with Copyright Protection

Hao-hua Chu
Department of Computer Science, University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, IL 61801, U.S.A.

Lintian Qiao
Department of Computer Science, University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, IL 61801, U.S.A.

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. We have implemented a prototype system which validates our secure multicast protocol and evaluated it against various performance matrices. The experimental results are very encouraging, but also show where new engineering approaches need to be deployed to conform fully to the design goals.

Keywords

Multicast security, copyright protection, key distribution, watermark

*This research is supported by National Science Foundation Career Grant NSF-CCR-96-23867, Research Board of University of Illinois at Urbana-Champaign and Air Force Grant, Number F30602-97-2-0121. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. For further author information, e-mail huawang,klara@uiuc.edu

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.

- *Multicast Architecture Independence*
  Our secure multicast protocol can be implemented on top of any existing multicast protocols: M-OSPF [54], DVMRP [54], CBT [6], or PIM [23]. 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.

- *Real-time Encryption*
  In order to provide secure data transmission, it is necessary to design encryption algorithms for multimedia data because of their special characteristics, such as their coding structure, large amount of data, and real-time constraints. In particular, we are interested in the secure algorithms for MPEG video streams. The MPEG video encryption algorithm should aim towards efficient and real-time processing so that they can become an integral part of the video delivery process and

at the same time preserve the highest security level and compression ratio.

- *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 leaked data is discovered and analyzed.

There are many works in group key distribution[2, 1], real-time video encryption[3, 32, 34, 45, 38], copyright protection[49, 47, 30, 28] and leakers identification[29, 14, 19]. We made the first attempt to integrate the problem of multicast key distribution, real-time video encryption, copyright protection, leakers identification in our previous work [20] by proposing a secure multicast protocol with copyright protection. The major contributions of this paper over our previous work are: (1) We extend the related work in the area of key distribution, real-time encryption and watermarking, (2) We modify the secure multicast protocol by using symmetric key cryptosystem instead of asymmetric key cryptosystem, (3) We apply and enhance the collusion-secure algorithm given by Boneh and Shaw[11] in our multicast watermark protocol to address the collusion problem while in our previous work we only give a brute force solution, (4) We propose a simple real-time encryption algorithm based on permutation operations, and (5) We add real testbed implementation and experimental results, which prove the feasibility of our secure multicast protocol.

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 presents our implementation and experiment; section 6 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. According to Rafaeli[40], group key distribution can be divided into three main categories: centralized approach, distributed subgroup approach and distributed approach. There is a large body of work in each category, such as the centralized approach in the Group Key Management Protocol(GKMP)[27, 26], Secure Lock[18], Hierarchical Binary Tree[55, 56], One-way Function Tree[4]; the distributed subgroup approach in Iolus[35], Intra-domain Group Key Management[25], Kronos[43], Marks[13]; and the distributed approach in Cliques[48], Octopus Protocol[9], Distributed Hierarchical Binary Tree,[41], Distributed Flat Table[55]. Below we will outline in detail some of the approaches, which have the most impact on our solution.

2.1.1 Core Based Tree Key Distribution

The Core Based Tree (CBT) key distribution by Ballardie [5] is based on the *hard state* multicast protocol like the Core Based Tree[6] 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

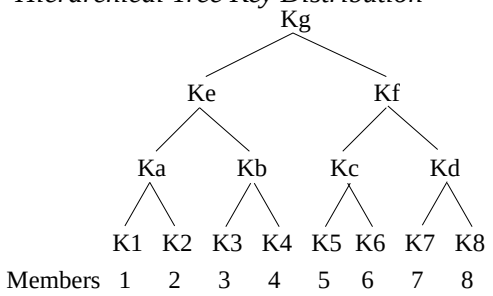

Figure 1: Hierarchical tree key distribution

The Hierarchical tree key distribution by Wallner [56] 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 ($\{K_a\}_{K_1}$[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. Some improvements can be found in the work by Balenson[4] et al, Canetti[15] et al, Perrig[36] et al, Banerjee[7] et al, and Hardjono [25, 24] et al.

2.1.3 Iolus

Iolus key distribution by Mittra [35] 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 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.

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

2.1.4 Cliques

Steiner[48] et al described Cliques, which is an extension to the Diffie-Hellman(DH) key agreement protocol. The multicast group agrees on a pairs of primes q and α such that α is primitive $mod\, q$. Each group member has a secret number x_i. The protocol consists of upflow and downflow stages. Assuming a group has n members, the first member calculates the first value α^{x_1} and passes it to the next member. In round i ($i \in [1, n-1]$) of the upflow stage, the ith member receives the set of intermediary values, raises them to its own secret number, generates a new set and unicasts to the $(i+1)$th user a collection of i values. Of these values, $i-1$ are intermediate and one is *cardinal*. The last member raises all intermediate values to its secret value. In the downflow stage, the last member multicasts the whole set of intermediate values. Each group member extracts its respective intermediate values and can easily calculate the key $k = \alpha^{x_1 x_2 \cdots x_n} \, mod\, q$.

This protocol achieves secure and efficient key agreement in the context of dynamic peer groups. However, the solution does not scale well to large groups since the size of upflow and downflow messages increases linearly, and the number of exponential operations also increases linearly with the size of group members.

2.1.5 MARKS

Briscoe[13] et al proposed MARKS. It slices the time-length of a multicast session into small portions of time and uses a different key for encrypting each slice. The encryption keys are leaves in a binary hash tree that is generated from a single seed. The key sequence is constructed as follows:

1. The sender randomly generates an initial state seed value $s(0, 0)$.

2. The sender decides on the required maximum tree depth D, which will lead to a maximum key sequence length $N_0 = 2^D$ before a new initial seed is required.

3. The sender generates two left and right first level intermediate seed values by applying respectively the left and right blinding functions to the initial seed.

4. The same algorithm is applied to the following levels until the expected depth is reached.

If a receiver is granted access from time slice m to time slice n, the sender unicasts a set of seeds to the receiver. The set consists of the intermediate seeds closest to the tree root that enable calculation of only the required range of keys, any key outside the range cannot be calculated. This protocol only works if the leaving time of a member is fixed when the member joins the group, and it can not be used in situations when a membership change requires change of the group key.

2.1.6 Kronos

Setia et al [43] proposed Kronos, which is based upon the idea of periodic group rekeying. They showed that in a large, highly dynamic group, the frequency of re-keying on each membership change becomes the primary bottleneck for scalable group re-keying. In Kronos, group re-keys are not driven by member join or leave requests. Instead, all the member join and leave requests that have accumulated during periodic intervals are processed, and the new multicast traffic encryption key is securely transmitted to the existing members of the group.

2.1.7 Our Approach

We have designed a secure multicast protocol with copyright protection which belongs to the centralized approaches. Our approach relies on a group leader that in synchrony with the sender (1) authenticates group members, (2) verifies the validity of messages, and (3) distributes message keys. More detailed discussion is presented in sections 3 and 4.

2.2 Real-time Video Encryption Issues

There already exist several encryption algorithms to secure MPEG video. The most straight forward method is to encrypt the entire MPEG stream using standard encryption methods. This is called the naive algorithm approach[3]. The greatest concern about this approach is the speed of processing during playback due to the large size of MPEG files.

Another method to secure MPEG streams is the selective encryption algorithm[32, 33], which encrypts only the I-frames of MPEG streams. Agi and Gong[3] have shown that great portions of the video are still visible partly because of interframe correlation and mainly from unencrypted I-blocks in the P and B frames. Therefore, selective method of only encrypting I frames may not work.

Meyer and Gadegast[34] have designed a MPEG-like bitstream SECMPEG that incorporates selective encryption and additional header information, and has high-speed software execution. But SECMPEG is not compatible with standard MPEG. A special encoder/decoder would be required to view unencrypted SECMPEG streams.

A proposal targeting at integration of compression and encryption of MPEG streams into one step is presented by Tang[51], where the basic idea is to use a random permutation list to replace the zig-zag order to map the individual 8x8 blocks to a 1x64 vector. This algorithm adds very little overhead to the video encryption and decryption process. But changing the zig-zag order to a random order will result in image size increase of about 25% to 60%, which is not tolerable because it defeats the purpose of compression.

Shi and Bhargava[45, 44] have developed fast MPEG video encryption algorithms which use a secret key randomly changing the sign bits of all DCT coefficients and the sign bits of motion vectors.

In one of our previous work Qiao and Nahrstedt[38] have developed the Video Encryption Algorithm(VEA), which utilizes the statistical behavior of MPEG compressed video and provides efficient, fast and secure encryption by encrypting one half of a frame with DES/IDEA, and the other half of the frame with "one-time-pad" generated from the frame.

In this paper, we use a simplified real-time video encryption algorithm which only involves permutation of the video data. It is much faster than DES and can be used in applications which do not require high security but have a strict real-time constraints.

2.3 Watermarking Issues

During the past few years, a number of digital watermarking methods has been proposed. Among the earliest works, L.F. Turner[52] 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[53] . There are many other proposals for watermarking such as Tanaka's schemes[50] which use the fact that the quantization noise is typically imperceptible to users, Brassil's methods[12] for textual document images, Caronni's geometric patterns (also called tags)[16], Steinbach, Dittmann and C. Vielhauer's PlataJanus for audio watermarking [47]. Some watermark algorithms on digital image and video can be found in the work of Zeng and Liu[57], Barni[8], Koch and Zhao[31], and Hartung[28], Suhail and Dawoud[49], Sridhar, Li and Nascimento[46], Kang, Kim and Han[30].

Craver, Memon, Yeo, and Yeung[22] address an important issue of rightful ownership. They provide an attack (*CMYY attack*) counterfeit watermarking scheme 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 [39] address and prove the non-invertibility property. Schemes applying to MPEG video stream and uncompressed video stream are designed.

In the multicast environment, the problem is that if all users share the same data with the same watermark, it is impossible to identify the user who leaked illegally data to the public.

Brown, Perkin and Crowcroft[14] proposed Watercasting, a distributed watermarking scheme of multicast media. In Watercasting, the source creates n differently watermarked copies of each media packet, where n is greater than the depth of the multicast group tree. Routers at the nodes in the multicast tree forward all but one of those packets out of each downstream interface on which there are receivers, and each last hop router forwards exactly one packet onto the subnet with the receiver. In this way, each receiver can have a stream with a unique combination of watermarked packets. Problem with this approach is that support in routers like reliable multicast is needed, and the log must keep the state of the entire network during the transmission.

Judge and Ammar[29] proposed watermarking multicast video with a hierarchy of intermediaries, which they called WHIM. It places a hierarchy of intermediaries as end systems in the network and forms an overlay network between them. Watermark is generated based on the receiver's location in the network and is inserted into the content incrementally as it traverses the network. Like Watercasting,

WHIM also adds some additional requirements to the network.

Chor et al.[19] proposed to multicast the same data, but to embed different encryption keys into the data. Each user would know where to look for his own key to decode the content. This idea was applied in their Traitor Tracing system for broadcast encryption. The problem with this approach is that some of the users can collaborate and change data in an unauthorized way. We call this problem the collusion problem, which was first addressed by Blakley et al[10]. Boneh and Shaw[11] give a collusion-secure fingerprinting scheme for digital data. They ask and answer the following question: if each user receives a different key of length m, and c users collude together to generate a new key, then how do we design the key set for all users such that based on the new key (created through collusion) we can find at least one member in the c-collusion group with error probability less than ϵ. Boneh and Shaw proved constructively that if the size of the multicast group is N and the size of the collusion group is c, then the generated key length must have the length of $O(c^4 log(N/\epsilon) log(1/\epsilon))$ to be c-secure with ϵ error. The problem with this approach is that we need to know the user group size N in advance to decide how many keys to generate. This is an issue in multicast where members can dynamically join and leave as we discuss in section 4. In our approach we apply the Boneh and Shaw's algorithm to the multicast multimedia environment and allow arbitrary user group size as shown in section 4.2.2.

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 a new member can join it, or it can accept and verify a payment information as a mean for a new member 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 requesting 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. The *JOIN* request contains the identity of the requesting member. We assume that both the requesting member and the group leader can properly authenticate each other in the secure unicast channel (e.g., via SSL) by presenting their certificates issued by well-known CAs (certificate authority) who authenticate their public keys. If the *JOIN* request is accepted, the group leader generates a unique member id *uid* that identifies the requesting member. The member id is then communicated to the requesting member. In addition to the unique member id, the multicast session needs to establish a symmetric key K_{uid} between the group leader and the requesting member *uid* for later use in the multicast session. If the secure unicast channel is setup using Diffie-Hellman, we can use the symmetric session key established between group leader and the requesting member *uid*. If the secure unicast channel is setup using RSA, the group leader would need to generate a symmetric key K_{uid} for the requesting member and transmits it to the new member. The member can now 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 keys cannot be predicted from the previous and subsequent generated keys.

The third component of the message contains the message key K_{msg} encrypted with the symmetric key K_{suid} between the group leader and the sender. K_{suid} is established between the sending group member and the group leader when the sending group member joins the multicast session through a secure unicast channel with the group leader. All three components are assembled into one data message. The sender multicasts the following data message in the multicast channel:

$$\{(suid, msgid), \{suid, msgid, data\}_{K_{msg}}, \{K_{msg}\}_{K_{suid}}\}$$

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_{suid} which can be used to decode K_{msg}.

Note that *suid* and *msgid* are repeated in the second component of the data message, so that an attacker cannot confuse the receivers by sending another message with the same $(suid, msgid)$ but with different data. These attacks are described in section 3.1.4.

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 looks up its current membership list to check that the sender is indeed a valid group member in (V1). Then it decrypts the third component of the data message using its symmetric key with sender K_{suid} to obtain the message key K_{msg} in (V2). It also decrypts the second component of the data message to verify that the $(suid, msgid)$ pairs are the same between the first component and the second component in the data message in (V3). This is needed to ensure that K_{msg} is indeed corresponding to message *msgid* from *suid*.

When both validation checks succeed, 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 symmetric key between the group leader and the corresponding member K_{uid} , 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).

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

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-R7). Upon receiving a data message, the receiver separates the data message into three components. 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 $\{suid, msgid, data\}_{K_{msg}}$ along with the message tag $(suid, msgid)$ in the first component as its lookup index in a data queue in (R2).

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 (R3-R4). 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 (R5). After the message key is decrypted, the receiver uses the message tag to retrieve the corresponding encrypted data component from the data queue in (R6). Then the receiver can decrypt the data with the message key in (R7). 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.

3.1.4 Attacks

To avoid computational overhead associated with digital signature and asymmetric encryption, data messages are symmetrically encrypted but not signed. This leads to some possible attacks by sending invalid messages to confuse receivers so that they cannot distinguish if a message is valid or not. We describe these attacks and how our protocol handles them.

In the first attack, an attacker pretends to be *suid*. The attacker multicasts an invalid data message with the same message tag $(suid, msgid)$ as a valid data message from a valid sender, but with a false $data'$, K'_{msg}, and K'_{suid}.

$$\{(suid, msgid), \{suid, msgid, data'\}_{K'_{msg}}, \{K'_{msg}\}_{K'_{suid}}\}$$

The group leader and group members would receive two data messages with the same message tag $(suid, msgid)$, a valid one from the sender *suid* and an invalid one from the attacker. Upon receiving the attacker's data message, the group leader follows steps in verification phase - it locates *suid* in the message tag, it applies the shared secret key K_{suid} to decrypt K'_{msg} from $\{K'_{msg}\}_{K'_{suid}}$, and then applies K'_{msg} to decrypt $\{suid, msgid, data'\}_{K'_{msg}}$. Since the attacker does not know K_{suid} (note that $K'_{suid} \neq K_{suid}$), the group leader gets garbage from decrypting $\{suid, msgid, data'\}$ which will not match the message tag. So the group leader would multicast an invalid verification message on $(suid, msgid)$. When the group leader receives the valid data message from sender *suid*, it also multicasts a valid verification message on $(suid, msgid)$. In other words, a group member receives two verification mes-

sages corresponding to the same $(suid, msgid)$, one for the attacker's data message and one for the valid data message. The question is how a group member can distinguish which data message is the valid one. Upon receiving a validation message, a group member follows the steps in the receiving phase - it locates the message key K_{msg}, and applies K_{msg} to decrypt $\{suid, msgid, data\}_{K_{msg}}$ (and $\{suid, msgid, data'\}_{K'_{msg}}$) from both valid and invalid data messages. The valid data message yields correct results with matching message tag, whereas the invalid data message yields garbage with incorrect message tag.

The first attack assumes that the attacker does not join the multicast session to become a valid group member, and the attacker cannot get K_{msg}. In the second attack, we assume that the attacker is a valid group member. After the attacker receives a verification message corresponding to $(suid, msgid)$, the attacker can obtain K_{msg}. Then the attacker multicasts the following invalid data message using the same message key K_{msg} as in the valid data message:

$$\{(suid, msgid), \{suid, msgid, data'\}_{K_{msg}}, \{K_{msg}\}_{K'_{suid}}\}$$

In the second attack, the attacker does not have K_{suid}. So the group leader can still distinguish if a message is valid or invalid. However, a receiver cannot do so, because both valid and invalid data messages have the same message key K_{msg}. A solution is that the group leader can detect if the attack comes from a valid group member or not, by checking if the attacker's message also has the same message key K_{msg} as the valid message. If the attack comes from a valid group member, the group leader multicasts an invalid-all message which tells receivers to discard all data messages with the message tag $(suid, msgid)$.

$$\{(suid, msgid), INVALID = ALL\}_{K^{gl}_{pri}}$$

The above solution does not solve the problem where an attacker can cause valid data messages to become invalid. A secure solution is that a sender signs all data messages using its private key, so that receiving group members and the group leader can verify the sender of data messages. It is more expensive computationally given that asymmetric encryption is involved.

$$\{(suid, msgid), \{data\}_{K_{msg}}, \{K_{msg}\}_{K_{suid}}\}_{K^{priv}_{suid}}$$

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 $(uid, \{K_{msg}\}_{K_{uid}})$ 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 message key slots corresponding to suspended members from 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 credentials (e.g. age) of members. The *SUSPEND* operation is very useful for a video-on-demand multicast where minors are temporarily suspended from seeing inappropriate materials.

Dynamic membership operations, including *JOIN*, *LEAVE*, *SUSPEND* and *EXPEL*, generate simple messages exchanged through a secure unicast channel between a member and its group leader. This secure unicast channel must ensure proper authentication of both the member and its group leader. This can be done by using a SSL connection where both the sender and the receiver authenticates each other through well-known CAs.

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[2], a lost message has no adverse effects on the future messages.

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

[2]In our implementation we apply a new key to a group of data messages, e.g. new key for every 300 frames. We do this to improve the network bandwidth overhead as described in section 3.3. If the key is lost, all 300 frames, corresponding to 10 seconds of video, are not decryptable, hence lost to the viewer. However, we believe that 10 seconds (even up to 40 seconds) of lost frames will not have major adverse effects on 1-2 hours of viewing/conferencing experience.

the multicast messages are available to everyone listening over the network.

3.3 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 multicasted data message. Let M be the number of key messages multicasted per second (or the $message\_rate$) and K be the size (in bits) of one slot in the third component of verification message ($(uid, \{K_{msg}\}_{K_{uid}})$ in section 3.1.2), the network bandwidth overhead is $O(N * M * K)$ bits.

The storage requirement at the group member consists of (1) the public keys of all 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 may not be true considering dynamic group membership. We compare our overhead with the Hierarchical Tree Key Distribution[56] in Table 1.

In the Hierarchical Tree protocol, the dominating overhead is in the rekey message which has a size $O(log(N)) * K'$ bits, where K' is the key size (in bits), N is the group size and $O(logN)$ is the number of keys a member holds. 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 $O(p * N * log(N) * K')$ bits. As for the storage overhead, each member needs to store the keys from its leaf to the root which is $O(log(N))$ bits.

For the network bandwidth overhead comparison between our protocol and the Hierarchical Tree protocol, it is $O(N * M * K)$ vs. $O(p * N * log(N) * K')$. Factoring out the common factor N and the constant K and K', it becomes M vs. $p * log(N)$. M is the message rate at which key is changed and it is usually a constant. For example, if a standard pay-per-view MPEG-2 video multicast runs at a message rate of 30 frames per second and key is changed per each data message, then $M = 30$. On the other hand, if key is changed only once a second, i.e., one per every 30 frames, then $M = 1$. In our implementation we change keys every 300 to 1200 frames to improve performance. In a multicast group where group members join and leave frequently, p can be assumed to be between 0.1 and 1. Given a reasonably large group size, e.g. $N = 1000$, $p = 0.1$, then $M = 1$ (change the key every 1 second) is as good as $p * log(N)$. The storage overhead is small in both protocols.

Another overhead in our protocol is the encryption time overhead which is the time spent in preparing the verification message. In a group of N members, the group leader uses N message key encryptions and 1 digital signature for each frame.

We can make a tradeoff between the network bandwidth overhead, the encryption time 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 shares the same message key for data encryption and decryption. Therefore, we prepare and multicast only one verification message every m data messages. This translates into m folds reduction in network bandwidth overhead and encryption time overhead with a tradeoff of t second(s) in *security vulnerability*. We define security vulnerability as follows:

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.

We calculate the approximate network bandwidth overhead and encryption time overhead, reusing the same message key on data messages within 40 seconds, for the group size ranging from 100 to 10000 for a MPEG-2 video multicast session in 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. If we use DES3 for encryption and RSA for signature, we estimate that it takes $1.6ms$ to encrypt a message key and the signature rate is 367KB/sec using a 512 bits long signature key[43, 17, 42]. Note that the bandwidth overhead ratio is computed as the network bandwidth overhead over the MPEG-2 bitrate, and the time overhead ratio is computed as the encryption time overhead over the message key reusing time. From Table 2, we can see that it is possible to realize the optimized version of our secure multicast protocol in real time for a reasonably large group.

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

Table 1: Overhead Comparison Between Our Key Distribution Protocol and the Hierarchical Tree Protocol

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

Table 2: Optimized Overhead for MPEG-2 Video Multicast

Group size	100	1,000	10,000
Network bandwidth overhead	0.36Kbps	3.6Kbps	36Kbps
Bandwidth overhead ratio	0.009%	0.09%	0.9%
Encryption time overhead	0.199s	1.99s	19.9s
Time overhead ratio	0.50%	4.98%	49.8%

watermarked copy by the the content provider. However, in the multicast environment, there is no way to differentiate among 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 extension to *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[3] also contains the two corresponding message keys, K_{msg}^{w0} and K_{msg}^{w1}, which are used to decrypt two watermarked frames.

$$\begin{aligned}\{(suid, msgid), \quad & \{suid, msgid, d_i^{w0}\}_{K_{msg}^{w0}} \\ & \{suid, msgid, d_i^{w1}\}_{K_{msg}^{w1}} \\ & \{K_{msg}^{w0}, K_{msg}^{w1}\}_{K_{suid}}\}\end{aligned}$$

4. During the verification phase, the group leader prepares the following augmented verification message [4] 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} but not the second watermarked frame d_i^{w1}.

$$\begin{aligned}\{(suid, msgid), \quad & VALID, \\ & (uid_1, \{b_{uid_1}^i, K_{msg}^{b_{uid_1}^i}\}_{K_{uid_1}}), \\ & (uid_2, \{b_{uid_2}^i, K_{msg}^{b_{uid_2}^i}\}_{K_{uid_2}}), .., \\ & (uid_n, \{b_{uid_n}^i, K_{msg}^{b_{uid_n}^i}\}_{K_{uid_n}})\}_{K_{gl}^{pri}}\end{aligned}$$

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:

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

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

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.

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

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 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 member 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 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 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 Detection

A collusion is defined as a behavior where more than one group member cooperate together to generate a new 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:

$$\begin{aligned}L = \quad & \{(uid_1, uid_2), (uid_1, uid_3), (uid_1, uid_4), \\ & (uid_2, uid_3), (uid_2, uid_4), (uid_3, uid_4)\}.\end{aligned}$$

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^{w1}_1. 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 c-combinations that cannot produce the

B_{leaked} bit value. The main question in the leaker identification problem is what is the minimum number of bits in the bit string per user to generate a c-collusion free solution. There exists a theoretical solution to this question as discussed in next subsection.

4.2.2 Collusion-free Selection

Let us consider the above example of four users, where two users collude and generate a new stream with new B_{leaked} bit string as shown in the following table. From this example, we can see that not all random bit strings B_{uid} can be used to uniquely identify collusion members.

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}	1	0	1	0	1

We can easily see that both (uid_1, uid_2) and (uid_3, uid_4) can be in the collusion group. In order to correctly identify the leakers, we need to use collusion-secure bit strings instead of randomly generating them as was first described in step 2 of our multicast protocol in section 4.1. Dan Boneh and James Show's collusion-secure fingerprinting scheme[11] can be used to generate collusion-secure bit strings. This solution presents a constructive algorithm to generate N $c-$secure bit strings with probability of detection error ϵ, where the length of each bit string is $O(c^4 log(N/\epsilon) log(1/\epsilon))$, c is the collusion group size, and N is the overall group size. One problem with Boneh's algorithm is that the $c-$secure bit string is too long for large group size and large collusion group. In the following table we give the lengths of collusion-secure bit string for different user group size N and collusion group size c with $\epsilon = 0.01$.

From Table 3, we can see that for two hour MPEG video stream that has 216,000 frames, the collusion-secure fingerprinting scheme[11] is only practical for a collusion group of size two.

In some applications with less strict security requirement, we can use a shorter collusion-secure bit string by allowing a larger detection error ϵ, as can be seen from Table 4. So if we are willing to tolerate a detection error of 10%, for a user group size of 100 and no more than three group members collude, we can generate a collusion-secure bit string which can be used in two hour MPEG video.

We can further decrease the length of collusion-secure bit string by using more than two differently watermarked video streams. As shown in Table 5, by using eight differently watermarked video streams, we can shorten the length of collusion-secure bit string to 1/3 of the length of that by using two eight differently watermarked video streams. Thus we are able to generate a collusion-secure bit string which can be used in one hour MPEG video for a user group size of 10,000 and collusion group size of three. The problem of this approach is that, in order to decrease the length of collusion-secure bit string to $1/n$ of that required in using two differently watermarked video streams, we need to use 2^n differently watermarked video streams.

From the above discussion, we can see that by adjusting the detection error ϵ and the number of differently watermarked video streams, we can generate practical collusion-secure bit string for small collusion group. Next, we discuss how to incorporate Boneh and Show's collusion-secure fingerprinting scheme into our secure multicast protocol.

The multicast process is divided into four parts: the Join Part, the Begin Part, the Critical Part and the End Part. Duration of each part is pre-decided by the group leader. The four parts division of a multicast process is very natural in the case of video-pay-per-view and video conference. For example, in video-pay-per-view, there is always a Join Part during which people can pay to watch the movie, a Begin Part during which some movie preview is aired, a Critical Part during which the movie is played, and an End Part.

Join Part	Begin part	Critical Part	End Part

In each part, the five steps of data transmission described in section 4.1 previously are used. The only difference in these parts is the generation of bit string B_{uid}. The group leader uses a random bit string B^{r1}_{uid} for the Join Part and the Begin Part, during which information of less importance and requiring less security is sent out and the group leader keeps track of the number of people in the multicast group. In the Join Part, people can join and leave the multicast group freely. In the Begin Part, people are not allowed to join the multicast group, but they are free to leave. Thus the group leader knows at most how many people are in the multicast group. Assuming collusion group of size two, the group leader generates a collusion secure bit string B^{s}_{uid} for each member (uid) using the collusion-secure fingerprinting scheme[11]. He also generates a random bit string B^{r2}_{uid} for each member (uid), the length of B^{s}_{uid} plus that of B^{r2}_{uid} equals the number of frames being multicasted in the Critical Part. The group leader concatenates these two bit strings and does a permutation P to the resulting string, generates a new bit string B^{sr}_{uid}. The permutation is kept as a secret of the group leader. The generation of B^{sr}_{uid} is done during the Begin Part and B^{sr}_{uid} is used in the Critical Part. Finally, in the End Part, the group leader uses a third random bit string B^{r3}_{uid}.

When an illegally leaked data stream is found, the group leader gets the B_{leaked} from the sender. From B_{leaked}, the group leader first extracts the substring B^{sr}_{uid}, corresponding to the Critical Part, then he does permutation P^{-1} to B^{sr}_{uid} and gets B^{s}_{uid} and B^{r2}_{uid}. By using Dan Boneh and James Shaw's Algorithm[11], at least one member in the collusion group can be identified from B^{s}_{uid}.

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 and the encryption overhead remain unchanged.

Table 3: Length of c-secure bit string for different user group size and collusion group size with $\epsilon = 0.01$ and 2 different watermarked streams

User group size	Collusion group size	Length of c-secure bit string(bits)	Time corresponding to MPEG video(min)
100	2	42,600	23.67
10,000	2	64,959	36.01
100	3	256,800	142.7
10,000	3	388,520	215.8
100	5	2,322,000	1290
10,000	5	3,489,984	1939

Table 4: Length of c-secure bit string for different user group size and collusion group size with $\epsilon = 0.1$ and 2 different watermarked streams

User group size	Collusion group size	Length of c-secure bit string(bits)	Time corresponding to MPEG video(min)
100	2	25,389	14.11
10,000	2	42,189	23.44
100	3	154,330	85.74
10,000	3	260,850	144.9
100	5	1,433,124	796.2
10,000	5	2,392,227	1329

5. IMPLEMENTATION AND EXPERIMENT

We have designed and implemented a system which validates our secure multicast protocol for video transmission. Furthermore, we have checked the system's feasibility, and evaluated it against various performance metrics such as real time execution. In this section, we describe our implementation and experiments.

There are two major pieces of our implementation - the multicast protocol and the secure multicast protocol that is layered on top of it. We have implemented a simple application-layer multicast protocol, based on an endpoint overlay multicast architecture, which is similar to the End-System Multicast[21]. Note that the focus of this paper is not on multicast protocols, but rather on a secure protocol layered on top of it. Hence, any simple and easy-to-implement multicast protocol would suffice for our validation.

5.1 Architecture

We make the following simplifications in our application-layer multicast protocol. We assume that there exists a Gateway node that can listen to new requests from member hosts who would like to join the multicast protocol. The location of the Gateway node is well-known to all member hosts. Since the multicast protocol can rely on the security multicast protocol layered on top of it for security, the Gateway simply accepts any host requesting to join the multicast session. Note that joining the multicast session is not the same as joining the secure multicast session. To join a secure multicast session, the member host would need to communicate with the group leader through a secure unicast channel to get a key and an id as described in section 3. Our end-system multicast protocol links the group members by unicast TCP connections with the Gateway as the relay center. A sender would first transmit data to the Gateway, and the Gateway forwards data to all members using unicast TCP connections. The use of per-member unicast TCP connections and a single Gateway is obviously inefficient, not robust, and not scalable. But it is simple to implement, and it is reliable so that we do not have to deal with data loss in video decoding.

The testbed is shown in Figure 3. The testbed contains 5 nodes linked together by unicast TCP connections with the Gateway as a relay center. The Group leader is responsible for authentication and key management as described in section 3. The member node is a normal member of the group and can JOIN/LEAVE at any time and SEND/RECEIVE data to/from the group. The member needs to know the location of the gateway in order to join the group. After it connects to the gateway, the gateway confirms from the group leader if the member can be a part of the group or not (by initiating the JOIN protocol). If verified, the member ends the JOIN protocol by sending its information to the group (via gateway). After joining successfully, the member enters the sending/receiving phase. It can either initiate a send session in which it can multicast either a pre-recorded video file (video-on-demand) or it can transmit live video in real-time, or it can become a receiver and receive video from another sender and display it on its screen in real-time.

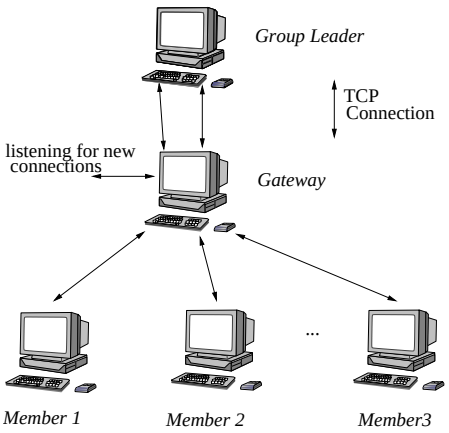

Figure 3: Architecture of the multicast protocol

Table 5: Length of c-secure bit string for different user group size and collusion group size with $\epsilon = 0.1$ and 8 different watermarked streams

User group size	Collusion group size	Length of c-secure bit string(bits)	Time corresponding to MPEG video(min)
100	2	8,463	4.703
10,000	2	14,060	7.813
100	3	51,443	28.58
10,000	3	86,950	48.30
100	5	477,708	265.4
10,000	5	797,409	443

5.2 Implementation Details

In this section, we discuss some implementation issues in detail. Since the multicast protocol is described in section 3, we will focus on implementation of our real time encryption algorithm, watermarking algorithms and network issues.

5.2.1 Real Time Encryption and Key Distribution

To transmit data securely between group members, video must be encrypted. On the other hand, to include the encryption/decryption process as part of the sending and receiving process, this process must be executed in real-time. We have first deployed the symmetric encryption algorithm DES; however, DES is very expensive and is more feasible for video streaming applications because it can not perform in real time, i.e., encrypt/decrypt 30 frames per second. Therefore, we have developed real-time MPEG video encryption algorithm[37] , which uses complex set of permutations and only encrypts half of the compressed video data. Thus we can achieve about 50% speed up over using DES to encrypt the whole compressed video data. In some applications, like news broadcasting, the video data only needs to be kept secret for a short time. In this case, we can use only simple permutations to achieve real-time performance and in the mean time to meet the moderate requirement of security. In this paper, we discuss the simple real-time permutation algorithm which is used in our implementation.

We use the concept of interleaving to disperse or permute the data. The data is divided into blocks of different sizes. The number of blocks and the size of each block depend on the key. We explain this with a simple example as shown in Figure 4:

Assume that the size of the data is 5000 bytes and the key with which it is to be encrypted is 5603417982. We first find the sum of the digits of the key. In this case it is 45. The total number of digits (including 0) is 10. So, we shall divide the data into 10 blocks(or 11, as we shall see). The sizes of the blocks will depend on the face value of each digit in the key sequence and it will be a percentage of the total file size. Hence the block sizes will be the following:

Block 1: (5/45) * 5000 = 555 B (5 is the first digit in the key 5603417982)

Block 2: (6/45) * 5000 = 666 B

Block 3: (0/45) * 5000 = 0 B (it exists only logically)

...

Block 9: (8/45) * 5000 = 888 B

Block 10: (2/45) * 5000 = 222 B

This will total to 4995 bytes since the sizes were truncated. It leaves an eleventh block of size 5. Once we have the block sizes, we start putting the data into blocks according to the following rule(as shown in Figure 4):

The next byte of the original data goes into the first available slot of the next block in the encrypted data. We maintain a pointer for each block which keeps track of the number of slots already used and the number of slots available in each block. By sequentially traversing through the original data, we pick up each successive byte and place it in the encrypted data by following the above rule. The last block of a few bytes (it can be proved that the size of this block will always be less than the number of digits in the key, 10 in this case) is copied at the end of the encrypted data without permuting. After this is done, we get the encrypted data which has the same size as the original data, but is permuted according to the key and exactly the same key is required to put it back in original form. The security analysis of the permutation encryption scheme is as follows:

1. We are using a 32 bit key and our implementation supports the scalability of any key length. Exactly the same key is required to decrypt the data. One encryption operation takes about 12 ms for a 7000 bytes image file, and to apply jpeg decoding, it takes about 13 ms. If someone tries to generate all the permutations of keys (there are 2^{32} possibilities) and decrypts the data by exhaustively applying all the keys (or in other words, by using brute force) , then on average, it will take about 621 days to break the key, and about 6.21 days if, say, we use 100 computers simultaneously by dividing the key space among them.

2. We change the keys after every 1000 frames, (approximately 30 seconds of video). So, the attacker code would need to be fast enough to break the code before that. If the code must need to be broken in say t seconds, then, one would require $\frac{2^{31}*25ms}{ts}$, or $\frac{53687091}{t}$ computers.

3. When the data is in the form of text, it is very easy to find patterns and to guess the key. But since we are dealing with binary data here (compressed image in the form of jpeg), no definite patterns can be found and therefore it is very difficult (almost impossible) to take a meaningful guess at the key.

4. It works even for a bitmap image since if we permute all the pixels in the image, we cannot make heads or tails of the new image. Hence, we can easily afford to use simple permutation of the data only, instead of use complex DES, assuming that the data will be video data.

The key distribution algorithm is described in sections 3.1 and 3.2. To achieve real-time key distribution, in our implementation, we use the same message key for data message with some fixed period of t seconds instead of using a different key for every frame. The tradeoff between the encryption time overhead and security is given in section 3.3 where we discuss security vulnerability.

5.2.2 Watermarking Issues and Leakers Identification

We attempted to design a real-time watermarking algorithm We started with implementing the algorithm proposed by Lintian Qiao et al.[39] but it can not run in real time. We also tried to embed some information in the compressed format, but for about 40% of the times the frames got distorted and the watermark became visible. Finally, we decided to embed the watermark in pixel domain, but this approach was way too slow. To our knowledge, there is no robust, invisible and noninvertible real time video watermarking algorithm. So in our implementation of the secure multicast protocol for live distributed multimedia applications, we do not include watermarking video frames. However, as a separate experiment, we have implemented the generation of the collusion-secure bit streams. We also simulate the creation of new bit streams through collusion of different group members and implemented Boneh and Show's algorithm for leaker identification. Our experiments show that we can successfully identify one of the leakers under various cases of collusion. This validates that our multicast watermark protocol can fulfill the goal of copy-right protection. In on-demand multimedia applications, if we do not require embedding of real-time watermark like in the case of video-on-demand, then we can watermark the video files off-line and our protocol can successfully integrate security and copy-right protection.

5.2.3 TCP vs UDP

We have used TCP for all the data transfers across the network in the protocol. Although it might be thought that UDP will be a better approach when dealing with large chunks of video data in a multimedia applications, we chose TCP for our implementation because:

1. TCP is a reliable protocol which means that there will be no bit-errors, no reordering or loss of packets. If we use UDP, our application has to take care of all these reliability issues in a real-time and efficient fashion.
2. Our main aim in this implementation is to analyze the feasibility of the secure multicast protocol in real-time applications. So, the main emphasis is on security issues in multicast group rather than the networking implementations and reliability. TCP alleviates us of taking care of the reliability and other networking issues and we can therefore concentrate on the important issues of multicast security.

5.3 Experiments and Results

5.3.1 Prototype Testbed

The testbed shown in Figure 3, consists of Sun Ultra-60 machines running SunOS, release 5.7. The machines are connected via 10/100 Mbps Ethernet cards. The experiments are designed to analyze the feasibility of the real-time secure multicast and to measure its performance against security by recording the net frame rate achieved in a real-time scenario.

5.3.2 Performance Metrics

In a real-time video application, the user expects a frame rate of about 25 to 30 frames per second. This gives us a processing time of about 33 ms to 40 ms for one frame. Our main aim in this implementation was to check if it is possible to actually record, watermark, encrypt, and send a video frame within this time period to achieve the desired *frame rate*. So, our main performance metric is the frame rate. This directly depends upon the time it takes to record and encrypt a single frame, which in turn depends on the size of the frame. Hence, we measure and discuss these processing times and metrics in the coming sections.

5.3.3 Experimental Scenarios

We perform our experiments under two different scenarios - video-on-demand, and, video-conferencing. For both types of experiments, a gateway and group leader are started to set up the group. Then three members join the group. The number of frames after which the key is changed is a variable for the different experiments. We use the motion JPEG format(MJPEG). The dimensions of the JPEG frames are 256x256 pixels, with a size of approximately 6 to 7 kilobytes per frame. The two scenarios are setup as follows:

- *Video-Conferencing*
  After the group setup, i.e., successful joining of group members, one machine represents the sender and the two other machines are the receivers. The live video is recorded from a camera attached to the sender. The sender records the video in realtime and sends it to the receivers after encrypting it. The experiment involves multicasting 5000 frames of video with an average frame size of about 6000 bytes.

  The number of frames after which the keys are changed is set at 300, 600, 900 and 1200 frames or approximately after period of 10s, 20s, 30s, and 40s of video, respectively. Table 6 shows the results of the four experiments together with the achieved frame rates:

  The readers should note that the average time to send one frame decreases considerably when the number of frames, after which the key is changed, increases. This is because in the time to send one frame we take into account (a) the time of the sender sends the data message, and (b) the time the group leader sends the verification message. As we increase the number of frames after which the keys are changed, we get better performance, but at the same time, we are weakening security. However, in a real world video broadcasting application, we shall normally be sending thousands of frames during 3 - 4 hours video conference sessions. We can easily afford to change keys after every 40 seconds

Table 6: Summary of performance results for the secure video-conferencing application

number of frames after which the key is changed	time to record one frame (ms)	time to encrypt one frame (ms)	time to send one frame (ms)	total time for processing one frame (ms)	net frame rate achieved (fps)
300	21.351	9.931	4.846	39.421	25.367
600	21.361	10.804	2.493	36.096	27.704
900	21.043	11.282	1.993	35.480	28.185
1200	22.909	9.615	1.441	34.715	28.806

of video for such applications. A frame rate of approximate 29 frames per second is still achieved which is in the desired range. Even if someone gets the keys for a particular range of video, by the time they decrypt the video, the keys would have changed.

The average time to record a frame from the camera is 21.666 ms which gives us approximately 11 to 12 ms for the rest of the processing. Average time to encrypt the data with our real-time encryption algorithm according to our algorithm described in section 3 is 10.408 ms. This data comprises of two copies of the original video frame or approximately 12,000 bytes. So, per byte it takes about 0.867 μs to encrypt. It takes on average 2.693 ms to send a frame across the LAN network. This time depends on the configuration of the network and also the load and congestion that the network is handling at that time. It is independent of the implementation of the protocol.

- *Video-On-Demand*
  After the group setup, i.e., successful joining of group members, one machine servers as the VOD server and the two other machines serve as VOD clients(see Figure 3). The video is pre-recorded and stored at the VOD server side. The server reads the data from the video file, frame by frame, and sends it encrypted to the group of VOD clients. The experiment involves multicasting a pre-recorded video file of exactly 5000 frames with an average frame size of approximately 7000 bytes.

  The number of frames after which the keys are changed is set at 300, 600, 900 and 1200 frames or approximately after period of 10s, 20s, 30s, and 40s of video, respectively. Table 7 shows the results of the four experiments along with the achieved frame rates:

  Again, similar results as in Table 6 can be observed. An increase in the number of frames after which the keys are changed, increases the performance. However, in a real world video application, the huge amount of video frames during a VOD session will allow for large intervals when keys will be changed.

The reader should note the significant increase of frame rate in VOD scenario when compared to video conferencing scenario. This is because the video is pre-recorded and compressed off-line. On average, it takes about 8.962 ms to read a compressed frame from a file, as compared to 21.666 ms (time to capture a frame in camera and compress it). The other values are comparable. For example, the encryption takes on average 11.300 ms per data frame which comprises of two copies of video frames of size 7000 bytes each. So, it takes about 0.807 μs to encrypt one byte. Average time to send one frame is 1.963 ms which is again independent of the protocol and instead depends on the network load at that particular time.

5.4 Lessons Learned

5.4.1 Quality of Service

In both scenarios, video conferencing and VOD, we have achieved the desired rate(25 fps and higher), our measured quality of service parameter, therefore we can conclude: (1) If video is pre-recorded and compressed off-line, we can achieve up to 40 fps and higher; (2) If live video is streamed, we can achieve up to 25 fps and higher; (3) Higher frame rates can be achieved if the interval between key changes is larger(e.g. 41fps in case of 300 frames interval versus 47 fps in case of 1200 frames interval in VOD scenario); (4) Copy right protection of the video data via watermarking can be only achieved for VOD application without violating QoS, as currently we can not achieve real-time watermarking during the capture-compress-encrypt process in the video conferencing application at the same time.

5.4.2 Data Encryption Schemes

We have initially started using the standard DES encryption scheme. Although this scheme is very secure, it is also very slow. However, our implementation is designed for real-time applications and expensive DES operations are not affordable. We needed a scheme which was as secure as DES when the data are binary JPEG files, and at the same time very fast. So, we came up with our own permutation encryption scheme described in section 5.2.1. It is much faster than the DES algorithm, it only permutes the data instead of permuting and substituting as done in DES, and it does not compromise security. The permutation encryption algorithm works well in real-time due to its speed, but it can be improved further to have higher security.

5.4.3 Multiple Sessions and Scalability

This implementation can handle only one video session at one time with limited number of users. This is because of the limitation of the gateway. It can be improved with multiple video sessions going on at the same time. The gateway can maintain a different list of sessions each comprising of the members involved in the session and the configuration of the sender. Furthermore, a tree structure for the gateway can be implemented comprising of parent and child gateways. This will take the load off the single gateway and greatly improve the performance of the system. However, note that our goal was to investigate and implement a prototype to validate the secure multicast protocol for a multimedia session with its integrated video streaming, real-time encryption, leak

Table 7: Summary of performance results for the secure video-on-demand application

number of frames after which the key is changed	time to read one frame from disk (ms)	time to encrypt one frame (ms)	time to send one frame (ms)	total time for processing one frame (ms)	net frame rate achieved (fps)
300	9.174	11.635	3.226	24.136	41.432
600	8.947	11.443	1.914	22.395	44.650
900	8.887	10.975	1.625	21.580	46.340
1200	8.839	11.145	1.086	21.160	47.259

identification and copy right protection. We do not aim at the scalability towards multiple sessions and multicast tree at this time. Therefore, we believe that we have achieved our objective and showed the possibilities and limitations of secure multicasting at the end systems.

5.4.4 Reliability

Our secure multicast protocol is based on a centralized group leader to periodically distribute keys and authenticate group members. This brings up a fault tolerance issue where the group leader becomes a single point of failure in the system. This problem can be solved by using replicas for the group leader node.

Reliability of the group leader is an important issue, however it was not our focus of research. We believe that with using standard distributed algorithms for fault tolerance such as *consistency protocols* between replicas of the group leader nodes, as well as *election protocols* to select a new group leader in case of failure of the active group leader, we can easily expand our secure multicast protocol and achieve reliability for our single point of failure.

6. 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 secure multicast watermark protocol. Both protocols do not require any security mechanisms in 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. We have validated our secure multicast protocol in the multimedia testbed, and tested it on a video conferencing and video-on-demand scenarios. The results in the live-video application show encouraging directions as we can consider real-time encryption as part of the capturing and displaying video process. The results in the VOD application are even better and show that if we perform copy-right protection of video data off-line, our protocol can successfully multicast video securely and with copy-right protection.

7. ACKNOWLEDGMENTS

We would like to thank Niti Yadav for her work in watermarking. We are also indebted to the anonymous referees for their invaluable comments.

8. ADDITIONAL AUTHORS

Additional authors: Hua Wang and Ritesh Jain

9. REFERENCES

[1] Msec working group. Available via http://www.securemulticast.org/msec-index.htm.

[2] The secure multicast research group. Available via http://www.securemulticast.org/smug-drafts.htm.

[3] I. Agi and L. Gong. An empirical study of Mpeg video trandmission. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, San Diego, CA, February 1996.

[4] D. Balenson, D. McGrew, and A. Sherman. Key management for large dynamic groups: One-way function trees and amortized initialization. Ietf draft, August 2000.

[5] A. Ballardie. Scalable multicast key distribution. Rfc1949, May 1996.

[6] A. Ballardie. Core based trees (CBT) multicast architecture. Rfc2201, September 1997.

[7] S. Banerjee and B. Bhattacharjee. Scalable secure group communication over ip multicast. In *Proceeding of ninth International Conference on Network Protocols*, Riverside, CA, November 2001.

[8] M. Barni, F. Bartolini, V. Cappellini, and A. Piva. Robust watermarking of still images for copyright protection. In *SPIE Proceedings '99, volume 3657*, pages 46–47, January 1999.

[9] C. Becker and U. Wille. Communication complexity of group key distribution. In *5th ACM Conference on Computer and Communication Security*, San Francisco, CA, November 1998.

[10] G. R. Blakley, C. Meadows, and G. B. Purdy. Fingerprinting long forgiving messages. In *Advances in Cryptology, Proceedings of CRYPTO '85, vol. 218 of Lecture Notes in Computer Science*, pages 180–189. Springer-Verlag, 1986.

[11] D. Boneh and J. Show. Collusion-secure fingerprinting for digital data. *IEEE Transactions on Information Theory*, 44(5):1897–1905, September 1998.

[12] 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*, volume 3, pages 1278–1287, Toronto, June 1994.

[13] B. Brisco and I. Fairman. Marks: Multicast key management using arbitrarily revealed key sequences. In *First International Workshop on Networked Group Communication*, November 1999.

[14] I. Brown, C. Perkins, and J. Crowcroft. Watercasting: Distributed Watermarking of Multicast Media. In *Proceedings of the First International Workshop on Networked Group Communication*, pages 286–300, Pisa, Italy, November 1999.

[15] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naorr, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of INFOCOM99*, March 1999.

[16] G. Caronni. Assuring Ownership Rights for Digital Images. In *Proceedings of Reliable IT Systems, VIS'95*. Vieweg Publishing Company, 1995.

[17] I. Chang, R. Engel, D. Kandlur, and D. Saha. A toolkit for secure internet multicast. Manuscript, 1998.

[18] G. H. Chiou and W. T. Chen. Secure broadcast using the secure lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, August 1989.

[19] B. A. Chor, A. Fiat, and M. Naor. Tracing traitors. In *Advances in Cryptology, Proceedings of CRYPTO '94, vol. 839 of Lecture Notes in Computer Science*, pages 257–270. Springer-Verlag, 1994.

[20] H. Chu, L. Qiao, and K. Nahrstedt. A secure multicast prorocol with copyright protection. In *Proceedings of IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*, San Jose, CA, January 1999.

[21] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM'01*, San Diego, CA, August 2001.

[22] 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*, volume 3022, pages 310–321, San Jose, CA, February 1997.

[23] 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*, volume 4, April 1996.

[24] T. Hardjono, B. Cain, and N. Doraswamy. A framework for grop key management for multicast security. Ietf internet draft(work in progess), August 2000.

[25] T. Hardjono, B. Patel, and M. Shah. Intra-domain group key management protocol. Ietf internet draft(work in progess), September 2000.

[26] H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Architecture. Rfc2094, July 1997.

[27] H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Specification. Rfc2093, July 1997.

[28] F. Hartung and B. Girod. Digital watermarking of uncompressed and compressed video. *Signal Processing(Special Issue on Watermarking)*, 66:283–302, May 1998.

[29] P. Judge and M. Ammar. Whim: Watermarking multicast video with a hierarchy of intermediaries. In *Proceedings of NOSSDAV2000*, Chapel Hill, NC, June 2000.

[30] H. Kang, K. Kim, and S. Han. Watermarking techniques using the drawing exchange format(dxf) file. In *Proceeding of ACM Multimedia 2001 Workshops*, Ottawa, Canada, October 2001.

[31] E. Koch and J. Zhao. Towards robust and hidden image copyright labeling. In *Proc. of 1995 IEEE workshop on Nonlinear Signal and Image Processing*, pages 452–455, Neos Marmaras, Greece, June 1995.

[32] Y. Li, Z. Chen, S. Tan, and R. Campbell. Security enhanced mpeg player. In *Proceedings of IEEE First International Workshop on Multimedia Software Development(MMSD'96)*, Berlin, Germany, March 1996.

[33] T. B. Maples and G. A. Spanos. Performance study of a selective encryption scheme for the security of networked, real-time video. In *Proceedings of 4th International Conference on Computer Communication and Network*, Las Vegas, Navada, September 1995.

[34] J. Meyer and F. Gadegast. Security mechanisms for multimedia data with the example mpeg-1 video. Available on www via http://www.powerweb.de/phade/phade.html.

[35] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[36] A. Perrig, D. Song, and J. D. Tygar. Elk, a new protocol for efficient large-group key distribution. In *2001 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 2001.

[37] L. Qiao and K. Nahrstedt. A New Algorithm for MPEG Video Encryption. In *Proceedings of The First International Conference on Imaging Science, Systems, and Technology (CISST'97)*, pages 21–29, Las Vegas, Nevada, July 1997.

[38] L. Qiao and K. Nahrstedt. Comparison of mpeg encryption algorithms. *International Journal on Computers and Graphics(special Issue: Data Security in Image Communication and Network)*, 22(3), January 1998.

[39] 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.

[40] S. Rafaeli. A decentralised architecture for group key management. PhD appraisal, Lancaster University, Lancaster, UK, September 2000.

[41] O. Rodeh, K. Birman, and D. Dolev. Optimized group rekey for group communication systems. Technical report, Hebrew University, 1999.

[42] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, 2nd edition, December 1995.

[43] S. Setia, S. Koussih, and S. Jajodia. Kronos: A scalable group re-keying approach for secure multicast. In *2000 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.

[44] C. Shi and B. Bhargava. An efficient mpeg video encryption algorithm. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, West Lafayette, Indiana, October 1998.

[45] C. Shi and B. Bhargava. A fast mpeg video encryption algorithm. In *Proceedings of the 6th ACM International Multimedia Conference*, Bristol, UK, September 1998.

[46] V. Sridhar, X. Li, and M. A. Nascimento. Towards robust hidden watermarking using multiple quasi-circles. In *Proceeding of ACM Multimedia 2001 Workshops*, Ottawa, Canada, October 2001.

[47] M. Steinbach, J. Dittmann, and C. Vielhauer. Platajanus: An audio annotation watermarking framework. In *Proceeding of ACM Multimedia 2001 Workshops*, Ottawa, Canada, October 2001.

[48] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transaction on Parallel and Distributed Systems*, 11(8):769–780, August 2000.

[49] M. A. Suhail and M. M. Dawoud. Watermarking security enhancement using filter parameterization in feature domain. In *Proceeding of ACM Multimedia 2001 Workshops*, Ottawa, Canada, October 2001.

[50] K. Tanaka, Y. Nakamura, and K. Matsui. Embedding Secret Information into a Dithered Multi-level Image. In *Proceedings of 1990 IEEE Military Communications Conference*, pages 216–220, 1990.

[51] L. Tang. Methods for encrypting and decrypting mpeg video data efficiently. In *Proceedings of The Fourth ACM Inernational Multimedia Conference(ACM Multimedia'96)*, Boston, MA, November 1996.

[52] L. F. Turner. Digital Data Security System. Patent IPN WO 89/08915, 1989.

[53] R. G. van Schyndel, A. Z. Tirkel, and C. F. Osborne. A Digital Watermark. In *Proceedings of the International Conference on Image Processing*, volume 2, pages 86–90, IEEE, 1994.

[54] D. Waitzman, S. Deering, and C. Partridge. Distance Vector Multicast Routing Protocol. Rfc1075, November 1988.

[55] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The versakey framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications(special Issue on Middleware)*, 17(9):1614–1631, September 1999.

[56] D. Wallner, E. Harder, and R. Agee. Key Management for Multicast: Issues and Architectures. Rfc2627, June 1999.

[57] W. Zeng and B. Liu. On resolving rightful ownships of digital images by invisible watermarks. In *IEEE International Conference on Image Processing*, volume 1, pages 552–555, Santa Barbara, CA, October 1997.

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.
K_{uid_i}, K_{suid} : symmetric key between $uid_i(suid)$ and the group leader.
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.

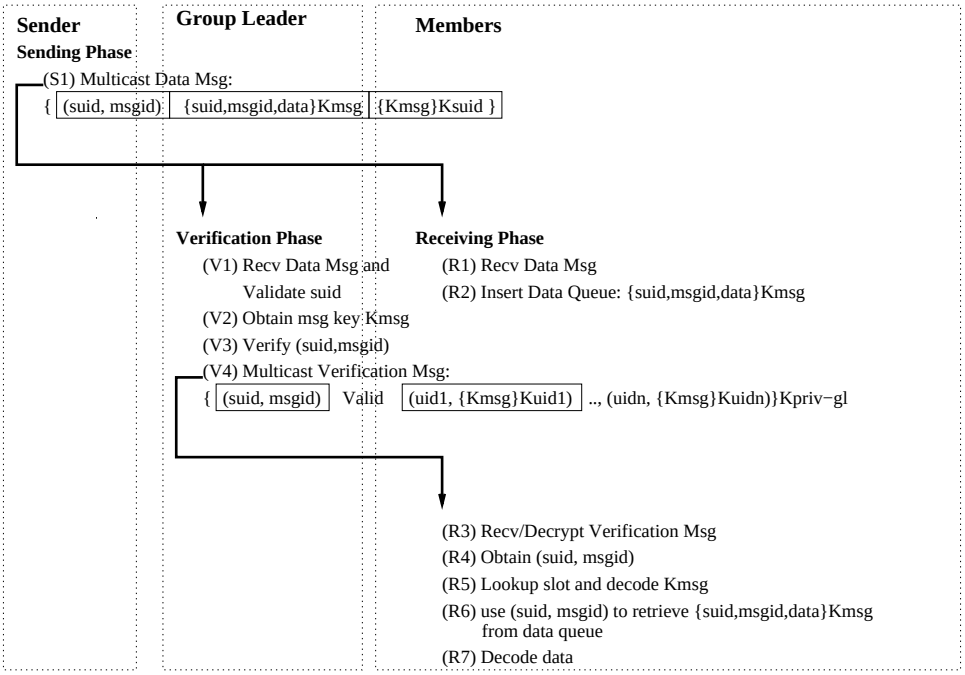

Figure 2: Three phases of our secure data transmission.

Size of Data File: 5000 Bytes,

Key: 5603417982

Sum of Digits: 45

Number of blocks: 10 + 1

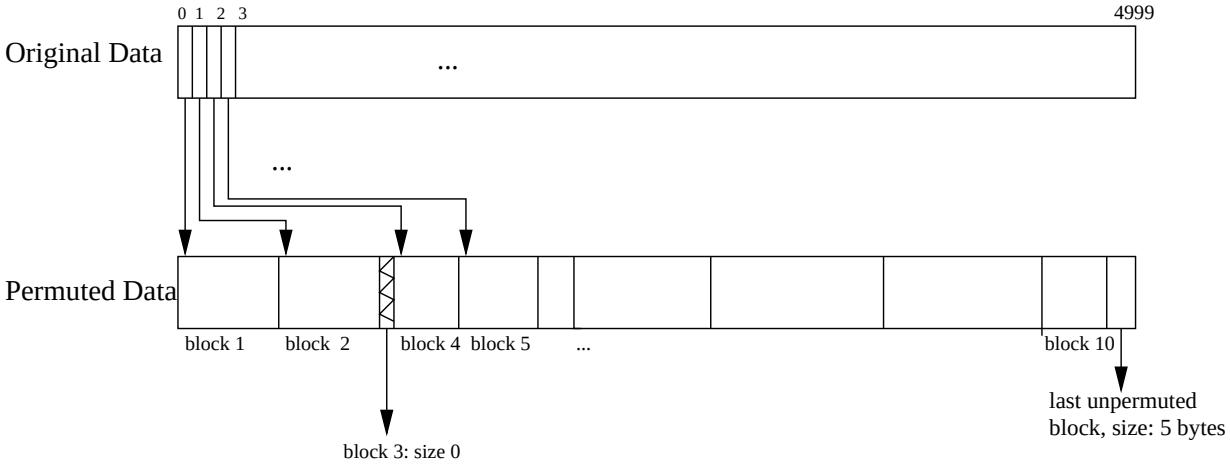

Figure 4: Permutation Encryption