A Distributed Group Key Management Scheme for Secure Many-to-many Communication

Lakshminath R. Dondeti, Sarit Mukherjee, Ashok Samal

Abstract— **Secure one-to-many multicasting has been a popular research area in the resent past. Secure many-to-many multicasting is becoming popular with applications such as private conferencing, distributed interactive simulation etc. Most of the existing secure multicasting protocols use a centralized group manager to enforce access control and for key distribution. In the presence of multiple senders it is desirable to delegate group control authority and key distribution tasks among the senders. We propose a distributed tree based key management scheme to support many-to-many group communication. Our protocol is scalable and places equal trust in all the senders. Key distribution overhead is evenly divided among the senders. We prove the correctness of the proposed protocol and provide a collusion analysis.**

I. Introduction

Secure multicasting in the Internet has several applications such as stock quote distribution, private conferencing, distributed interactive simulation etc. Some of these applications have a single sender distributing secret data to a large number of users while the others have a large number of users communicating privately with each other. Several protocols [1], [2], [3], [4], [5], [6], [7], [8], [9] have been proposed in the recent past to support group communication between one sender and several members. Very few solutions exist to facilitate secure communication between many senders.

Multicasting is a scalable solution to group communication and several existing secure one-to-many multicasting protocols are scalable as well [10]. Many-to-many secure multicasting protocols must also be scalable. Group access control, secret key distribution and dynamic group management are three major components of a secure group communication protocol. Most of the existing one-to-many secure multicast protocols use a centralized entity to enforce access control and distribute secret keys. When the multicast group membership is dynamic, the group manager must also maintain perfect forward secrecy. This is to guarantee that members cannot decrypt secret data sent before they join the group and the data sent after they left [11]. The group manager changes the appropriate secret keys when a member joins or leaves, and distributes them to the corresponding members. Rekeying process must be scalable, i.e., key distribution overhead should be independent of the size of the multicast group.

Although it presents a single point of attack and failure, using a centralized entity for group control is natural for one-to-many secure multicasting. However, in the presence of multiple senders, it is desirable that the multicast group be operational as long as at least one sender is operational. In other words, many-to-many secure multicasting calls for decentralized control of the group. Access control, key distribution and dynamic group management tasks must be delegated to all the senders. It is desirable to evenly distribute access control responsibilities and protocol processing overhead among all the senders in the group.

Only a few secure many-to-many group communication protocols exists in the literature. Secure multicast protocols which distribute session key(s) before sending secret data may be used for many-to-many group communication. However, all such protocols in the literature [1], [4], [5], [7], [9] use centralized group control and thus are prone to single point of attack as well as failure. Iolus [2] claims support for multiple senders. However, it exposes secret keys to third party entities which assist in key distribution and it also uses a centralized "group security controller (GSC)" for group management. The distributed flat key management (DFKM) scheme presented by Waldvogel et. al [3], [9] suggests the idea of placing equal trust in all the group members. Members joining early generate the keys and distribute them to the members joining late. While DFKM works in principle, it is susceptible to collusions. It is possible to have a very small subset of members controlling the group in DFKM, allowing uneven distribution of group control and key distribution overhead.

In this paper, we present a scalable secure multicasting protocol that supports many-to-many communication.

Lakshminath R. Dondeti is with the Department of Computer Science and Engineering, 115 Ferguson Hall, University of Nebraska-Lincoln, Lincoln, Nebraska, 68588-0115, email: ldondeti@cse.unl.edu (Corresponding Author)

Sarit Mukherjee is with the Panasonic Information & Networking Technology Laboratories, 2 Research Way, Princeton, NJ 08540, email: sarit@research.panasonic.com

Ashok Samal is with the Department of Computer Science and Engineering, 115 Ferguson Hall, University of Nebraska-Lincoln, Lincoln, NE, 68588-0115, email: samal@cse.unl.edu

Each member is assigned a binary ID and these IDs are used to define *key associations* for each member. Members in the key association groups are contacted to report membership changes and to exchange keys. All members are trusted equally and all of them may be senders. Prospective members may contact any active members to join the group. Active members verify new members' credentials and assign them a unique ID. The ID assignment is done locally without any need to lookup a global space of IDs. The ID assignment process illustrates the distributed nature of our protocol. The member which assigns a new ID is required to initiate the rekeying process. Note that rekeying is done to ensure perfect forward secrecy. Leaves are processed similar to joins; the neighbor (neighbors are determined based on IDs) of the departing host is required to notice the departure and initiate the rekeying process. Key associations help delegate key distribution overhead evenly among all the members of the group.

Members are represented by the leaves of a key distribution tree which is strictly binary. Each member generates a unique secret key for itself and each internal node key is computed as a function of the secret keys of its two children. All secret keys are associated with their blinded versions, which are computed using a one-way function [12], [13], [5]. Each member holds all the unblinded keys of nodes that are in its path to the root and the blinded keys of nodes that are siblings of the nodes in its path to the root. Contribution of a unique secret toward the computation of the root key gives each member partial control over the group. A join/leave requires only the keys in the path to the root from the joining/departing host to be changed. Thus, each membership change necessitates only $O(\log n)$ messages where n is the number of members in the group. Thus our protocol is scalable as well.

The rest of the paper is organized as follows. Section II defines the components and requirements of a secure many-to-many group communication protocol. Section III provides the description of the distributed group key management scheme proposed by us. We also prove its correctness and provide a collusion analysis of the protocol. Section IV summarizes existing secure multicast protocols that support multiple senders. We list our conclusions in Section V.

II. Components and Requirements of a Secure Many to Many Multicasting Protocol

Any secure group communication protocol has three major components, viz., group access control, secret key distribution and dynamic group management. Senders are responsible for controlling access to the secure multicast group. All members' authentication must be verified before they can join the group. Data is encrypted for privacy reasons before being sent to the group. The senders are responsible for distributing these data encryption keys to members in a secure and scalable fashion. Finally, the senders are responsible for maintaining perfect forward secrecy [11]. To ensure perfect forward secrecy, sender(s) must change secret keys when a host join or leaves the group. This rekeying process must be secure as well as scalable.

We now summarize the requirements and desirable characteristics of a secure many-to-many protocol. A secure group communication scheme must be scalable. More specifically, key distribution overhead must be scalable as the number of members (or senders) in the group increases. All senders must be trusted equally and the group must be operational if at least one sender is operational. It is desirable to distribute access control and dynamic group management tasks to all senders. This allows the join and leaves to be processed locally, thus avoiding global flooding of control traffic. Distribution of group management tasks also avoids performance bottlenecks [14] and eliminates single points of attack in a multicast group. Finally, the protocol must be able to avoid or detect and eliminate any colluding members or senders efficiently.

III. A Distributed Key Management Scheme using One Way Function Trees

We now propose a distributed tree-based key management scheme (DTKM) for secure many-to-many group communication. Our protocol is scalable and it trusts all members equally. Any or all of the members may be senders. In the rest of this discussion, we use the words members and senders interchangeably. DTKM delegates group control responsibilities and key distribution tasks evenly to all the members.

Members of the multicast group are represented by leaf nodes of a key distribution tree (see Figure 1). The key distribution tree is strictly binary, i.e., each internal node has exactly two children. Each member generates a unique secret key which is the member's contribution towards the generation of the internal node keys including the root key. Internal nodes are associated with secret keys and these keys are computed as a function of their children's keys. The root key is computed similarly and is used for data encryption. For each secret key there is a blinded version, which is computed by applying a given one-way function [5], [12] to the secret key. Given a blinded key, it is computationally infeasible to compute its unblinded counterpart. Each member knows all the keys of the nodes in its path to the root of the tree and the blinded keys of siblings of the nodes in its path to the root of the tree and

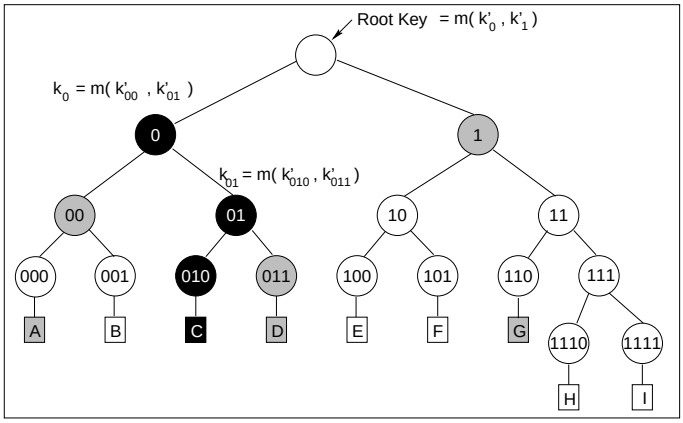

Fig. 1. An Example Key Distribution Tree: k' is computed from k using a one-way function[2] and m is a mixing function.

no other blinded or unblinded keys [5]. The blinded keys are distributed by members that are owners and authorized distributors of those keys. Each member computes the unblinded keys of the internal nodes of the tree in its path to the root and the root key itself, using the blinded keys it receives and its own secret key. A mixing function [5] is used to compute internal node keys from the blinded keys of the node's children.

Each node is assigned a binary ID and is responsible for generating a secret key. It also computes the blinded version of its key and shares it with its immediate neighbor in the key distribution tree. The following Find_Neighbor() algorithm [3] takes a binary ID $B = b_h b_{h-1} \dots b_1$, where $b_i, 1 \leq i \leq h$ is a binary digit, as its input and returns the binary ID of B's neighbor, B_n.

Algorithm 1: Discovering the Neighbor

```
Find_Neighbor(B = b_h b_{h-1} ... b_1)
begin
if(leaf_node(B_n = b_h b_{h-1} ... b_1bar) == "true")
  return B_n;
else if(internal_node(B_n) == "true")
  do
    B_n = B_n 0;
  while(leaf_node(B_n) == "false");
return B_n;
end
```

Following the above algorithm, H(1110)'s neighbor is I(1111), and G(110)'s neighbor is H(1110) in Figure 1. Neighbors with IDs of same length (H and I in Figure 1) are referred to as immediate neighbors and they exchange blinded versions of their secret keys with each other. If a pair of neighbors have different ID lengths (G and H in Figure 1), the member with the smaller ID size, sends the blinded version of its secret key and receives the blinded key of the corresponding internal node of same ID length from the member with the larger ID length (G receives k'_{111} from H). Using the new keys received the members compute their parent's secret key. A mixing function (typically XOR function) [5] is used to compute internal node keys. For example in Figure 1, C and D apply the mixing function m, to the blinded keys k'_{010} and k'_{011} to compute the internal node key k_{01}.

Keys are exchanged between members of a key association group in DTKM. Key associations are designed to delegate the task of key distribution evenly among all the senders. Each member needs as many blinded keys as the length of its ID, to compute the root key. Each blinded key is supplied by a different member of its key association group. For each bit position in a member's ID, there exists a member that supplies the corresponding blinded key. The following algorithm Find_Key_Association(), returns the ID of the member and the secret key it supplies, corresponding to a given bit position in a member's ID.

Algorithm 2: Finding the Members of a Key Association

```
Find_Key_Association(B = b_h b_{h-1} ... b_1, i)
begin
B_i = b_h b_{h-1} ... b_{i+1} bar(b_i) b_{i-1} ... b_2 b_1;
if((leaf_node(B_i) == "true")
  return B_i;
else if(internal_node(B_i) == "true")
  do
    B_i = B_i 0;
  while(leaf_node(B_i) == "false");
return B_i;
else
  do
    B_i = right_shift(B_i,1));
  while(leaf_node(B_i) == "false");
return B_i;
end
```

We illustrate the key association algorithm applied to H(1110) in Figure 1. Complementing the corresponding bit positions 1, 2, 3 and 4, we get I (1111), 1100, 1010, 0110. Since nodes with the last three IDs do not exist, we right-shift them by one bit position to get G(110), F(101) and D(011) as the rest of the members in H's key association group.

Next, we demonstrate the root key computation process for C(010). First, C generates the key k_{010} and sends its blinded version k'_{010} (computed using the given one-way function) to D(011). Similarly, D sends k'_{011} to C. Both

[2] k' is the blinded counterpart of the secret key, k

[3] We use several simple functions in the algorithms presented in this paper. Their definitions are listed in Appendix A.

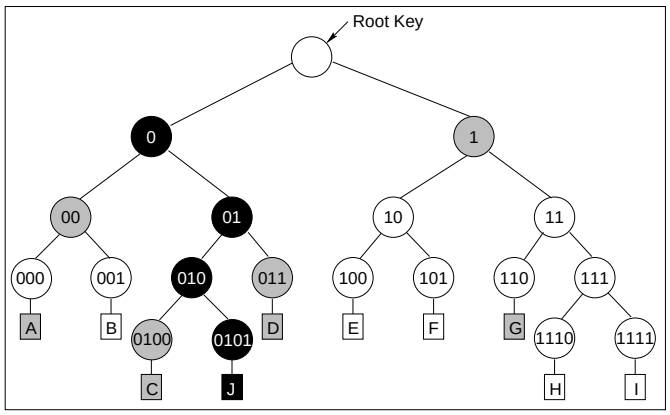

Fig. 2. Join Protocol

C and D can then individually compute the KEK k_{01} by applying the given mixing function to k'_{010} and k'_{011}. Next, C sends k'_{01} to A(000) and receives k'_{00} in return. After the key exchange, both A and C can compute k_0. After this step, C and G exchange k'_0 and k'_1 with each other. The root key is computed as a function of k'_0 and k'_1. Following similar steps, each member of the multicast group acquires or computes k'_0 and k'_1 and then computes the root key. All keys are encrypted with the recipient's public key before transmission. Note that C receives only the blinded keys of the siblings of the nodes in its path to the root. Using those keys, it can compute the unblinded keys of the nodes in its path to the root.

A. Join Protocol

Any or all members of the multicast group may be senders and are equally trusted in our scheme. A prospective member may join at any node of the key distribution tree. It is desirable to keep the key tree balanced for efficiency. However, that is only possible if one or more entities keep a snap shot of key distribution tree. To keep track of all members of the group and their positions in the key tree, we need either member status report messages broadcast to the whole group or a centralized entity that keep tracks of all joins and leaves. The first approach creates excessive network traffic and the second has a single point of failure. Our protocol attempts to locally balance the tree by choosing members in the tree that are within an administratively or Time-to-Live (TTL) scoped area. Prospective members join at a local member of the multicast group with the smallest ID length.

In Figure 2, J is a new member which joins at C. Upon verifying J's credentials [8], C splits its ID 010 (shown in Figure 1), keeps 0100 for itself and assigns 0101 to J. C also changes its secret key and sends the blinded version of its new key to J. J generates a secret key of its own and transmits the blinded version to C. Note that all keys corresponding to the internal nodes in the path to the root from J, change due to the join. J needs all the unblinded keys of the nodes shown in black and the blinded keys of the nodes show in gray, in Figure 2. Notice that none of the blinded keys known to C have changed and thus it can compute all the new keys corresponding to nodes 010, 01 and 0 and the root key once it receives k'_X. Now J needs the blinded keys corresponding to 011, 00 and 1. Using the Find_Key_Association() algorithm presented earlier, it determines that nodes with IDs 011(D), 000(A) and 110(G) are the members of its key association group. Note that these nodes and their neighbors also need the blinded keys that J knows or can compute. To elaborate, J sends k'_{010} to D and receives k'_{011} from D. It then computes k'_{01} and sends it to A and receives k'_{00} in return. A is also required to locally multicast k'_{01} encrypted with k_{00}, which can only be decrypted by A and B. J can now compute k'_0 which it sends to G, receives k'_1 in return and computes the root key for itself. G multicasts k'_0 encrypted with k_1, to be decrypted by E, F, G, H and I only. After the above key exchanges all authorized members will have the keys they need to compute the new root key. In all, there will be $O(\log n)$ unicast messages and $O(\log n)$ subgroup multicast messages during a join. Note that the multicast messages will be limited to a TTL-scoped or administratively scoped region, since they only need to be sent to selected subgroups within the multicast group. We now generalize the join process in the following Join() algorithm. It takes the new member and an exisiting member's ID as arguments.

Algorithm 3: Joining the Multicast Group

Join(X, $Y = b_h b_{h-1} \ldots b_1$) /* Y is the existing member */
begin
$X = b_h b_{h-1} \ldots b_1 1$;
$Y = b_h b_{h-1} \ldots b_1 0$;
$i = 1$;
while($i \leq$ length(X))
  begin
  member_id = Find_Key_Association(X, i);
  outgoing_key $= k'_{\text{right\_shift}(X, i-1)}$;
  send_key_from_to(outgoing_key, X, member_id);
  scoped_secure_multicast(outgoing_key, member_id);
  correction = length(X) − length(member_id);
  incoming_key $= k'_{\text{right\_shift(member\_id}, i-1-\text{correction})}$;
  send_key_from_to(incoming_key, member_id, X);
  $i = i + 1$;
  $k_{\text{right\_shift}(X, i-1)}$ = m(outgoing_key, incoming_key);
  end
end

B. Leave Protocol

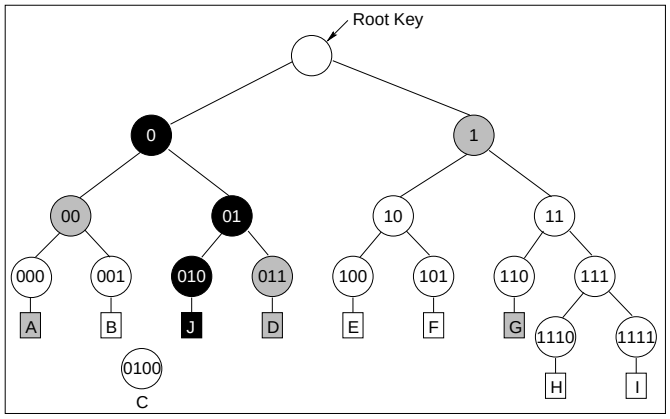

Fig. 3. Leave Protocol

When a member leaves, its neighbor initiates the rekeying process. If the neighbor is the departing member's sibling, it assumes its parent's position in the key distribution tree. Otherwise it notifies the descendants of the departing member's sibling to change their IDs. In either case, the neighbor changes its secret key and initiates the rekeying process. It sends the new keys to the members of its key association group and they are responsible for propagating the new keys to the appropriate members in their subgroups. In the rest of this section, we describe the ID update process followed by the rekeying process.

Assuming that X is the departing node and Y (= Neighbor(X)) is its neighbor, if Y has the same ID length as X, Y right shifts its ID by one bit position to get its new ID. If Y's ID is longer than that of X, X's sibling and its descendants change their IDs as follows. Notice that each descendant Z of X's sibling shares a key with X. If Z= $b_h b_{h-1} \ldots b_{i+1} b_i b_{i-1} \ldots b_2 b_1$, then Z's ID after the departure would be $b_h b_{h-1} \ldots b_{i+1} b_{i-1} \ldots b_2 b_1$, where i is the difference in the length of X's and Z's IDs plus one. In both cases, Y generates the new secret key and initiates the rekeying. In Figure 3, if E leaves, F gets the ID 10 and generates a new secret key; if G leaves, H and I get the IDs 110, 111 respectively and H generates the new secret key.

In Figure 3, C leaves the multicast group. J notices the departure and changes its ID from 0101 to 010, and generates a new secret key for itself. Consequently, internal node keys on J's path to the root change and J is responsible for initiating key exchanges with its counterparts, 011(D), 000(A) and 110(G) as defined earlier in this section. J sends the blinded key k'_{010} to D. Both J and D can now compute k_{01}. J then sends k'_{01} to A, which is responsible for sharing it with all members with the KEK k_{00}. Finally, J sends k'_0 to G which in turn sends k'_0 to all the members that have the KEK k_1. Notice that J does not need any keys in return from D, A or G. It already has the blinded keys it needs to compute the root key. While the departing member C knows all those blinded keys as well, it does not know any unblinded keys it needs and thus cannot compute or acquire the root key. A departure results in $O(\log n)$ unicast messages and $O(\log n)$ multicast messages, each message carrying one encrypted secret key. In the following, we provide a generalization of the rekeying process after a member departs from the group.

Algorithm 4: Leaving the Multicast Group

```
Leave(X)
begin
Y = Find_Neighbor(X);
if(length(Y) > length(X))
  foreach Z in (descendants(sibling(X))
    Z = delete_i^th_bit(Z, length(X)-length(Z)+1);
else if (length(Y) == length(X))
  Y = right_shift(Y, 1);
k_y = generate_new_key();
compute_internal_node_keys(Y);
i = 1;
while (i ≤ length(Y))
  begin
  member_id = Find_Key_Association(Y, i);
  outgoing_key = k'_{right_shift(Y,i-1)};
  send_key_from_to(outgoing_key, Y, member_id);
  scoped_secure_multicast(outgoing_key, member_id);
  i = i + 1;
  end
end
```

C. Secure Data Communication

All members in the multicast group can compute the root key with the given keys. A member with data to send, encrypts it with the root key and sends it via traditional multicast channels (eg: MBONE). Other members can decrypt the data without any further key exchanges.

D. Correctness of the Protocol

McGrew and Sherman [13] provide a detailed description of the properties of one-way functions and mixing functions that can be used for secure key distribution. One may use similar functions in implementing DTKM as well. The following system invariant that ensures the secrecy of the system also was defined in [13]. In this section, we prove that DTKM does not violate the system invariant in distributing secret keys.

System Invariant [5], [13]: Each member has all the unblinded keys corresponding to the nodes in its path to the root and the blinded keys of the siblings of the nodes in its

path to the root and no other blinded or unblinded keys.

Theorem 1 (Correctness) DTKM distributes all necessary keys to the corresponding members without violating the system invariant defined above.

Proof: Steps in the proof:

- A given host X generates its own secret key (unblinded key) and receives exactly $O(\log n)$ blinded keys. It computes the internal node keys (unblinded keys).
- Which unblinded keys do I get from unicasts?
- Which keys do I get from subgroup multicasts?
- In each case, see if I can get any more than I should. Do I violate the system invariant?

■

E. Collusion Analysis

The collusion analysis is to ensure that a subset of hosts cannot use their combined knowledge to extract secret information that any one of the individual members of the subset does not already know [15]. These hosts may be active members of the multicast group or members that were once part of the multicast group. First, we list some collusion scenarios and prove that DTKM is secure against those collusions. Adversary $\mathcal{A}1$ is the set of all members that were once part of the multicast group. Adversary $\mathcal{A}2$ is any subset of active members. Adversary $\mathcal{A}3$ is the union of $\mathcal{A}1$ and $\mathcal{A}2$.

Theorem 2: DTKM is secure against attacks from adversaries $\mathcal{A}1$, $\mathcal{A}2$, and $\mathcal{A}3$.

Proof: We prove DTKM's resilience towards collusions in three parts, each part addressing one of the adversaries separately.

Case $\mathcal{A}1$: $\mathcal{A}1$ is a set of all entities which were once members of the group. The blinded keys known to a member that just departed are not changed at the time of its departure. That is the only residual knowledge of the member that leaves the group. We examine the combined knowledge of members that departed from the multicast group. $\mathcal{L}$ and $\mathcal{R}$ denote the set of all such members that departed from the left subtree(s) and the right subtree(s) of the key distribution tree respectively. Without loss of generality we can assume that elements of $\mathcal{L}$ leave first. Their combined knowledge consists of several blinded node keys from the right subtree(s) of the key distribution tree. But, when any member of the corresponding right subtree(s) leave, their unblinded node keys and thus the corresponding blinded node keys change and the keys known to $\mathcal{L}$ are not active anymore. $\mathcal{A}1$'s knowledge is now limited to $\mathcal{R}$'s knowledge. Therefore, $\mathcal{A}1$ is not a successful collusion.

Case $\mathcal{A}2$: $\mathcal{A}2$ is a proper subset of active member set. Let h be the only active member that does not belong to $\mathcal{A}2$. The collective knowledge of the members of $\mathcal{A}2$ consists of all the blinded and unblinded node keys in the multicast group except h's own secret key (unblinded key). Since that key is independently generated by h and is not shared with anyone, members of $\mathcal{A}2$ cannot obtain the key, the only secret information of the group that they do not already know. Also, the one-way function makes it computationally infeasible to derive h's unblinded key from its blinded counterpart, which $\mathcal{A}2$ knows. Hence, $\mathcal{A}2$ is not a successful collusion either.

Case $\mathcal{A}3$: Adversary $\mathcal{A}3$ is the union of $\mathcal{A}1$ and $\mathcal{A}2$. From the "system invariant" all active members have access to unblinded keys of the nodes in their path to the root and the corresponding blinded keys and no other unblinded or blinded keys. Although, members of $\mathcal{A}1$ may have access to some blinded keys that members of $\mathcal{A}2$ do not have access to, the collusion does not produce any new information since the unblinded keys corresponding to the additional blinded keys are not available.

■

IV. Previous Work

Secure one-to-many multicast protocols [1], [4], [5], [7], [9] that predistribute the session key may claim that they support multiple senders as well. However, they use a centralized group controller which is a single point of attack. In the presence of multiple senders, it is desirable to have the group operational as long as at least one of the senders is operational. Group control and key management tasks should be evenly distributed among all the senders.

Only a few secure many-to-many group communication protocols exists in the literature [2], [9]. Iolus [2] uses hierarchical subgrouping to delegate group control authority as well as key distribution overhead. A group security controller, which is a centralized entity is assigned the responsibility of the security and operation of the group. Also, Iolus exposes secret keys to "trusted" third parties, which is a liability to the security of the system. Waldvogel et. al. proposed a distributed flat key management scheme (DFKM) which trusts all the members equally. In principle it conforms to the desirable characteristics of the group. However, it is possible in this protocol to have only a few senders controlling the operation of the group. This scheme cannot avoid collusions and it is hard to detect them as well. Finally, eliminating colluding senders could result in the re-initiation of the entire group. In the following, we describe Iolus and DFKM in detail.

A. Centralized Key Management (CKM) Schemes

In the centralized schemes, a group manager distributes the secret keys to the members. The sender or a trusted

third party entity acts as the group manager. Most centralized schemes use a key distribution tree [1], [4], [5], [9] to distribute the keys, while others (flat-schemes) [7], [9] use binary IDs of members as a reference for key distribution. In the tree-based schemes, members are represented by the leaf nodes of the tree. Each node of the tree is associated with a secret key and each member receives all the keys in its path to the root. A join or a leave requires all the keys in the path from the joining/departing node's position to the root to be changed. The group manager is responsible for distributing the new keys secretly to appropriate members. The protocol proposed by McGrew et. al [13] uses one-way functions to compute internal node keys, thereby reducing the number of messages required to update keys during a join or a leave. The flat schemes propose the use of $2W$ key encrypting keys(KEK), where W represents the length of an ID and a group key. There is a key corresponding to each value (0 or 1) of the bit positions. The group manager assigns IDs to members and distributes appropriate keys securely. Group manager is also responsible for key updates during group membership changes. The flat schemes use only $O(\log n)$ keys, however they cannot avoid, detect or eliminate collusions efficiently.

The group key may be used by any member to encrypt secret data and thus CKM schemes support many-to-many multicasting. However, they have a single point of control, attack and failure in the group manager. The group manager also could be a performance bottleneck as well [14]. While it is acceptable to use a centralized entity for group management in one-to-many communication, it is desirable to delegate group control authority as well as key distribution overhead evenly among all the senders in many-to-many secure multicasting.

B. Iolus

Iolus [2] proposes the idea of hierarchical subgrouping for scalable secure multicasting. Group security agents (GSA) share a secret key with each of their subgroup members. Similarly a group security controller (GSC) distributes a secret key to the top level subgroup. All these keys serve the purpose of KEKs while session keys are distributed during multicast data transfers. GSAs forward the session keys to their members. When a host joins, the corresponding GSA changes its subgroup key and distributes the updated key to all the subgroup members including the joining host. When a member leaves, the GSA distributes the new key to all the subgroup members except the departing host. For secret data distribution, the sender generates a session key, encrypts the data and sends it via multicast channels. It then encrypts the session key with its subgroup key and sends it to the top level subgroup. The GSAs decrypt the session key, encrypt it with their subgroup key and forward it to their subgroup members. Notice that any member in the group may send data and then send the session key(s) to its GSA, which propagates the session key(s).

GSAs are typically third party routers and Iolus does not provide any mechanism to hide secret data from them. While the use of GSAs make Iolus scalable, it is a serious limitation for a secure key distribution protocol to trust third party entities. GSC is a centralized entity and presents a single point of failure. An adversary may take over the operation of the GSC and disrupt the protocol.

C. Distributed Flat Key Management Scheme (DFKM)

In DFKM [9], each member is assigned a unique binary ID. There are $2W$ key encrypting keys(KEK), where W represents the length of an ID, in the multicast group. There is a key corresponding to each value (0 or 1) of the bit positions. There is no group manager in this scheme. Each member of the multicast group is trusted and holds/creates the traffic encrypting key (or the session key) and W KEKs corresponding to its ID. Members joining early generate the keys and are called *key holders*. The ones joining late, receive keys from these key holders. Each key holder is required to announce its keys so that joining members can determine the state of the group. New members receive the session key and the KEKs corresponding to their IDs. If any KEK is not already in the group, the new member creates it and announces itself as the key holder for that key. The protocol also presents mechanisms to resolve synchronization problems in key generation also. For example, if two different members generate the same key, all recipients of those keys merge the two keys to generate a new key instead of exchanging additional control messages to resolve ownership rights.

DFKM uses $O(\log n)$ keys to provide secure communication. While it may be efficient to use as few keys as possible, it allows a subset of the members of the multicast group to have access to all keys in the group. In DFKM it is possible to have as few as two hosts taking control over the group (members with IDs $000\ldots0$ and $111\ldots1$ for example). This scheme cannot avoid such collusions and it is hard to detect them as well. It may be necessary to re-initiate the group to eliminate the colluding senders and re-initialization is rather costly.

Table I compares the protocols that support many-to-many group communication. In the table, CKM represents the centralized tree based key distribution schemes. Note that while the flat schemes use less keys, they cannot avoid, detect or eliminate collusions efficiently.

TABLE I
COMPARISON OF SECURE MANY-TO-MANY MULTICAST PROTOCOLS

Criterion	CKM	Iolus	DFKM	**DTKM**
Group control	Centralized	Distributed	Distributed	Distributed
Single point of failure	Group Manager	Group Security Controller	No	No
Vulnerable to collusions	No	No	Yes	No
Uses trusted third party entities	No	Yes	No	No
Equal distribution of control	No	No	Not always	Yes
No. of keys in the group	$O(n)$	$O(n)$	$O(\log n)$	$O(n)$
No. of keys at a member	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$
No. of messages during a join	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$
No. of messages during a leave	$O(\log n)$	$O($Average size of a subgroup$)$	$O(\log n)$	$O(\log n)$
Scalable	Yes	Yes	Yes	Yes

V. CONCLUSIONS

One-way function trees have been used in the cryptography literature for various purposes. Recently McGrew and Sherman [13] proposed the idea of using bottom-up one-way function trees (OFT) for secret key distribution in large dynamic groups. DTKM uses one-way functions, but proposes a distributed solution to group key management. In the following we summarize our contributions.

Distributed ID Assignment DTKM proposes a localized ID assignment scheme thereby eliminating the need for a centralized group controller. We introduce the idea of key associations, which facilitate the delegation of group management functions as well as key distribution overhead to all the senders.

Distributed Group Management Each member generates its own key thereby contributing a secret towards the computation of the root key. This property gives each member equal control over the group. It also ensures that no proper subset of the group members can gain control of all the blinded and unblinded keys in the group. In OFT, the group manager needs to monitor all members of the group to detect unscheduled departures. In DTKM neighbors monitor each other thereby avoiding global flooding of control traffic (eg: heart-beat messages).

Immunity to Collusions DTKM does not have a single point of control, attack or failure. We guarantee that no proper subset of the senders can gain access to all the keys in the multicast group, this avoiding collusions. If a subset of the senders are compromised they can be eliminated without having to re-initiate the group.

Scalability DTKM supports secure group communication between a large number of senders in a scalable fashion.

REFERENCES

[1] D. Wallner, E. Harder, and R. Agee, "Key Management for Multicast: Issues and Architecture," IETF Draft, July 1997.

[2] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting," in *Proc. ACM SIGCOMM*, Cannes, France, September 1997, pp. 277–288.

[3] G. Caronni, M. Waldvogel, D. Sun, and B. Plattner, "Efficient Security for Large and Dynamic Groups," Tech. Rep. TIK Technical Report No. 41, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, February 1998.

[4] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," in *Proc. ACM SIGCOMM*, August 1998.

[5] D. Balenson, D. McGrew, and A. Sherman, "Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization," IETF Draft: draft-balenson-groupkeymgmt-oft-00.txt, Feb 1999.

[6] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast Security: A Taxonomy and Efficient Constructions," in *IEEE INFOCOM*, New York, March 1999.

[7] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha, "Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques," in *IEEE INFOCOM*, New York, March 1999.

[8] L. R. Dondeti, S. Mukherjee, and A. Samal, "A dual encryption protocol for scalable secure multicasting," in *Fourth International Symposium on Computers and Communications*, Red Sea, Egypt, July 1999.

[9] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The VersaKey Framework: Versatile Group Key Management,"

JSAC Special Issue on Networked Multimedia (to appear), August 1999.

[10] L. R. Dondeti, S. Mukherjee, and A. Samal, "Survey and comparison of secure group communication protocols," Tech. Rep., University of Nebraska-Lincoln, June 1999, Submitted for Publication.

[11] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press series on discrete mathematics and its applications. CRC Press, 1997.

[12] A. Fiat and M. Naor, "Broadcast Encryption," in *Advances in Cryptology: Proceedings of Crypto 1993*, 1993, pp. 480–491, LNCS 773.

[13] D. A. McGrew and A. T. Sherman, "Key establishment in large dynamic groups using one-way function trees," *Submitted to IEEE Transactions on Software Engineering*, May 1998.

[14] L. R. Dondeti, S. Mukherjee, and A. Samal, "Secure group communication using dual encryption: its security and performance analysis," Tech. Rep. UNL-CSE-1999-001, University of Nebraska-Lincoln, February 1999, Submitted to JSAC Special Issue on Network Security.

[15] S. H. Low, N. F. Maxemchuk, and S. Paul, "Anonymous credit cards and their collusion analysis," *IEEE/ACM Transactions on Networking*, December 1996.

APPENDIX

A. GLOSSARY

Function definitions used in this paper:

- leaf_node(X) returns true if X is a leaf node of the key distribution tree; false otherwise.
- internal_node(X) returns true if X is an internal node of the key distribution tree; false otherwise.
- right_shift(B, i), takes a binary ID $B = b_h b_{h-1} \ldots b_2, b_1$ and a number, i as its inputs and right shifts B for i time(s). The output will be $b_h b_{h-1} \ldots b_{i+1}$.
- send_key_from_to(key, X, Y) indicates that X sends "key" to Y.
- scoped_secure_multicast(key, X) indicates that X encrypts the "key," and locally multicasts it.
- length(X) returns the number of bits in the binary ID X.
- generate_new_key() returns a new secret key.
- compute_internal_node_keys(Y) indicates that Y locally computes all internal node keys and their blinded counterparts.
- m() is the mixing function.
- If $X = b_h b_{h-1} \ldots b_2 b_1$, sibling$(X) = b_h b_{h-1} \ldots b_2 \overline{b_1}$.
- descendants(X) returns the members of the multicast group that are descendants of X.
- sibling(X) returns the sibling of node X.
- delete_i^{th}_bit(B, i), takes a binary ID and an integer as its inputs and returns B with its bit position i deleted. For example if $B = b_h b_{h-1} \ldots b_{i+1} b_i b_{i-1} \ldots b_2 b_1$, the function returns $b_h b_{h-1} \ldots b_{i+1} b_{i-1} \ldots b_2 b_1$.