2000

DISEC: A Distributed Framework for Scalable Secure Many-to-Many Communication

Lakshminath R. Dondeti
University of Nebraska-Lincoln

Sarit Mukherjee
Panasonic Information & Networking Technology Lab

Ashok Samal
University of Nebraska-Lincoln, asamal1@unl.edu

DISEC: A Distributed Framework for Scalable Secure Many-to-many Communication

Lakshminath R. Dondeti
Carrier IP Evolution Group, Nortel Networks
600 Technology Park Drive, MS E65-60-202
Billerica, MA 01821
ldondeti@nortelnetworks.com

Sarit Mukherjee
Panasonic Information &
Networking Technology Lab
2 Research Way, Princeton, NJ 08540
sarit@research.panasonic.com

Ashok Samal
University of Nebraska-Lincoln
115 Ferguson Hall, Lincoln, NE 68588-0115
samal@cse.unl.edu

Abstract

Secure one-to-many multicasting has been a popular research area in the recent past. Secure many-to-many multicasting is becoming popular with applications such as private conferencing and distributed interactive simulation. 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 management responsibility to all the senders. We propose a distributed group key management scheme to support secure many-to-many communication. We divide key distribution overhead evenly among the senders. Our protocol is scalable and places equal trust in all the senders.

1. Introduction

Secure multicasting in the Internet has several applications such as stock quote distribution, private conferencing and distributed interactive simulation. 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, 4, 3, 8, 10, 11, 12] have been proposed in the recent past to support group communication between one sender and several members. Most of them use a centralized entity for group management. In the presence of multiple senders, distributed group management is desirable. Existing solutions advocate distributed group management, but delegate key distribution overhead unevenly [9, 10]. We present a scalable group key management scheme for secure many-to-many communication that distributes overhead evenly among the members of a multicast group.

Although it presents a single point of attack and failure, using a centralized entity for group management 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. Key distribution and dynamic group management tasks must be delegated to all the senders. It is desirable to evenly distribute protocol processing overhead among all the senders in the group.

Only a few distributed group management protocols for scalable secure many-to-many communication exist in the literature [8, 9, 10]. Iolus [8] can 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 [10] 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 also possible to have a very small subset of members controlling the group in DFKM, allowing uneven distribution of group control and key distribution overhead. Rodeh et. al [9] also advocate distributed group management for efficiency. However, their protocol assigns group control and management to a subset of the members rather than to all of them.

We present DISEC, a distributed framework for scalable secure many-to-many communication. In this paper, we de-

0-7695-0722-0/00 $10.00 © 2000 IEEE

scribe the key distribution mechanism in DISEC. We assign binary IDs to members and define a *key association* for each member based on its ID. Members forward secret keys through their key association groups during rekeying. Prospective members may contact any active member 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 new member initiates the rekeying process. Note that rekeying is done to ensure perfect forward secrecy [7]. When a member leaves, its neighbor (neighbor of a member is also determined based on the member's ID) notices the departure and initiate the rekeying process. Key associations help delegate key distribution overhead evenly among all the members of the group.

We use a virtual binary key distribution tree to better explain our scheme. The leaves of the tree represent members of a multicast group. Each member generates a unique secret key for itself. Each internal node key is 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 [1, 5, 6]. 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.

We organize the rest of the paper as follows. Section 2 provides a description of DISEC. Section 3 compares DISEC to existing secure multicast protocols that support multiple senders. We list our conclusions in Section 4.

2. A dissection of DISEC

We propose a distributed group key management scheme for scalable secure many-to-many communication. All of the members are senders. Therefore, in the rest of this discussion, we use the terms members and senders interchangeably. DISEC delegates group control responsibilities and key distribution tasks evenly to all the members. Our protocol is scalable and it trusts all members equally. In the following, we concentrate on the key distribution mechanism of DISEC. Details of group access control are out of the scope of this paper.

For better explanation of our protocol, we represent the members of a multicast group by leaf nodes of a virtual binary key distribution tree (see Figure[1] 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 represent secret keys and they 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 [1, 5] 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 no other blinded or unblinded keys [1]. 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. We use a mixing function [1] to compute internal node keys from the blinded keys of the node's children.

The limited distribution of unblinded and blinded keys is to ensure that DISEC is immune to collusions. It can be proven that no proper subset of the senders can gain access to all the keys in the multicast group. Collusion analysis and proof are out of the scope of this paper.

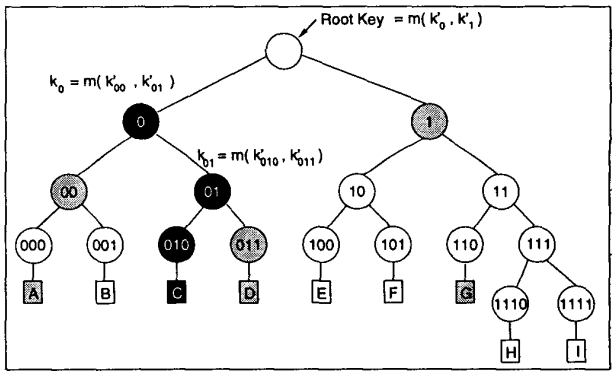

Figure 1. An example key distribution tree: we use a one-way function to compute k' from k and m is a mixing function.

Each member has 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 Find_Neighbor() algorithm[2] (Algorithm 1) takes a binary ID $X = b_h b_{h-1} \ldots b_1$, where $b_i, 1 \leq i \leq h$ is a binary digit, as its input and returns the binary ID of X's neighbor, X'.

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

[2] We use several simple functions in the algorithms presented in this paper. They are self explanatory.

Following Algorithm 1, 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. We use a mixing function (typically XOR function) [1] 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}.

Algorithm 1: Discovering the neighbor

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

2.1. Key association groups

Algorithm 2: Finding members of a key association group

```
Find_Key_Association(X = b_h b_{h-1} ... b_1, i)
begin
    X_i = b_h b_{h-1} ... b_{i+1} (bar)b_i b_{i-1} ... b_2 b_1;
    k_i = k_{b_h b_{h-1} ... b_{i+1} (bar)b_i};
    if (leaf_node(X_i) == "true")
        return(X_i, k'_i);
    else if (internal_node(X_i) == "true")
        do
            X_i = X_i0;
        while (leaf_node(X_i) == "false");
        return(X_i, k'_i);
    else
        do
            X_i = right_shift(X_i,1));
        while (leaf_node(X_i) == "false");
        return(X_i, k'_i);
end
```

To help delegate the task of key distribution evenly among all the members, we define key association groups. Members of a key association group exchange keys in DISEC. Each member needs as many blinded keys as the length of its ID, to compute the root key. For each bit position in a member's ID, there exists a member that supplies the corresponding blinded key. We call the set of members that supplies blinded keys to a member, a key association group. The Find_Key_Association() algorithm returns the ID of the member and the secret key it supplies, corresponding to a given bit position in a member's ID.

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. I, G, F and D supply the keys k'_{1111}, k'_{110}, k'_{10}, k'_0 respectively, to H.

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 C and D can then individually compute 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. We compute the root key from the keys 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. 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.

2.2. Join protocol

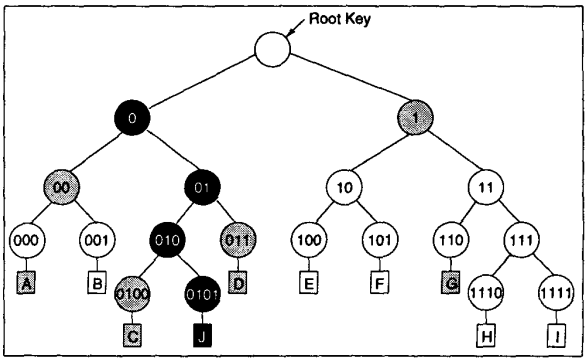

Figure 2. Join protocol

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 keeps track 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, during joins. Prospective members join at a local member of the multicast group with the smallest ID length.

Algorithm 3: Joining the multicast group

```
Join(X, Y = b_h b_{h-1} ... b_1) /* Y is the existing member */
begin
    Y = b_h b_{h-1} ... b_1 0;
    X = b_h b_{h-1} ... b_1 1;
    k_x = generate_new_key();
    i = 1;
    while (i ≤ length(X))
    begin
        (M, κ') = Find_Key_Association(X, i);
        outgoing_key = k'_{right_shift(X,i-1)};
        send_key_from_to(outgoing_key, X, M);
        scoped_secure_multicast(outgoing_key, M, κ);
        send_key_from_to(κ', M, X);
        i = i + 1;
        k_{right_shift(X,i-1)} = m(outgoing_key, κ');
    end
end
```

In Figure 2, J is a new member which joins at C. To include J into the group, 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'_J. 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 generalize the join process in Algorithm 3. It takes the new member and an existing member's ID as arguments. In the algorithm, κ' indicates the key sent by M to X.

2.3. 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.

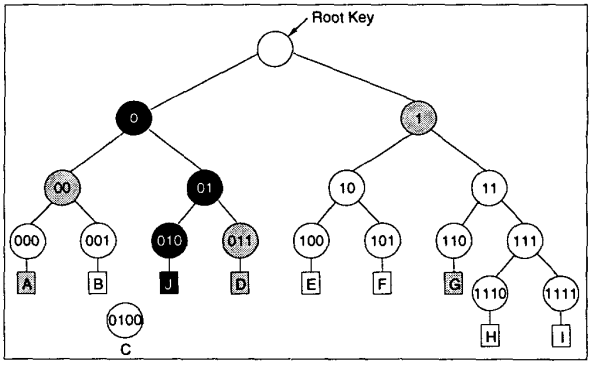

Figure 3. Leave protocol

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 Z's and X'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 gen-

erates 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 who have k_{00}. Finally, J sends k'_0 to G which in turn sends k'_0 to all the members that have 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);
    foreach Z in {descendants(sibling(X))} ∪ {Y}
        Z = delete_i^th_bit(Z, length(Z)−length(X)+1);
    k_y = generate_new_key();
    compute_internal_node_keys(Y);
    i = 1;
    while (i ≤ length(Y))
    begin
        (M, κ') = Find_Key_Association(Y, i);
        outgoing_key = k'_right_shift(Y,i−1);
        send_key_from_to(outgoing_key, Y, M);
        scoped_secure_multicast(outgoing_key, M, κ);
        /* M already has κ */
        i = i + 1;
    end
end
```

2.4. 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. Our protocol allows secure subgroup communication also. A sender may send secret data to a subgroup of members by encrypting the key it shares with the subgroup.

3. DISEC compared to existing solutions for secure many-to-many group communication

Secure one-to-many multicast protocols [1, 3, 10, 11, 12] that pre-distribute 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 failure and 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 distributed group management protocols for secure many-to-many communication exist in the literature [8, 9, 10]. Iolus [8] uses hierarchical subgrouping to delegate group control authority as well as key distribution overhead. A group security controller, 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 distributed group management. However, it is possible in this protocol to have only a few senders controlling the operation of the group. Also, this scheme cannot avoid collusions and it is hard to detect them as well. Eliminating colluding senders could require re-initiation of the entire group. Recently, Rodeh et. al [9] proposed a distributed group key management scheme using a logical hierarchy of keys (DLKH). This protocol assigns group control and management to a subset of the members, to avoid a single point of failure and attack.

Table 1 compares the protocols that support many-to-many communication. In the table, CKM represents the centralized key distribution schemes. Note that while the flat schemes use less keys, they cannot avoid, detect or eliminate collusions efficiently.

4. Conclusion

We propose a distributed key management scheme for many-to-many secure group communication. We use one-way function trees for key distribution and management. One-way function trees have been used in the cryptography literature for various purposes. Recently McGrew and Sherman [6] proposed the idea of using bottom-up one-way function trees (OFT) for secret key distribution in large dynamic groups. DISEC uses one-way functions, but proposes a distributed solution to group key management. In the following we summarize our contributions.

- **Distributed ID assignment:** DISEC proposes a localized ID assignment scheme thereby eliminating the

Table 1. Comparison of secure many-to-many multicast protocols

Criterion	CKM	Iolus	DFKM	DLKH	DISEC
Group control	Centralized	GSAs Distributed GSC Centralized	Distributed	Distributed	Distributed
Single point of failure	Group Manager	GSC	No	No	No
Vulnerable to collusions	Tree-based No Flat Yes	No	Yes	No	No
Uses trusted third party entities	No	Yes	No	No	No
Even distribution of control	No	No	Not always	No	Yes
No. of keys in the group	Tree-based $O(n)$ Flat $O(\log n)$	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$
No. of keys at a member	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$	$(O(\log n)$
No. of messages during a join	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
No. of messages during a leave	$O(\log n)$	O(Average size of a subgroup)	$O(\log n)$	$O(\log n)$	$O(\log n)$
Scalable	Yes	Yes	Yes	Yes	Yes

need for a centralized group controller. We introduce the idea of key associations, which facilitates 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 DISEC, neighbors monitor each other thereby avoiding global flooding of control traffic (example: heart-beat messages).

- **Scalability:** DISEC supports secure group communication between a large number of senders in a scalable fashion. Key distribution overhead is distributed evenly among all the senders. Key associations ensure that each sender serves only $O(\log n)$ other senders which contributes to the scalability of the protocol.

DISEC is a structured protocol in that each sender can determine the other senders it serves and the keys they need without having to exchange any control information.

References

[1] 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.

[2] 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.

[3] 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.

[4] L. R. Dondeti, S. Mukherjee, and A. Samal. A Dual Encryption Protocol for Scalable Secure Multicasting. In *Fourth IEEE Symposium on Computers and Communications*, Red Sea, Egypt, July 1999.

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

[6] 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.

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

[8] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *Proc. ACM SIGCOMM*, pages 277–288, Cannes, France, September 1997.

[9] O. Rodeh, K. Birman, and D. Dolev. Optimized group rekey for group communication systems. In *Network and Distributed System Security Symposium*, San Diego, CA, February 3-4 2000.

[10] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The VersaKey Framework: Versatile Group Key Management. *JSAC Special Issue on Service Enabling Platforms For Networked Multimedia Systems*, August 1999.

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

[12] C. K. Wong, M. Gouda, and S. S. Lam. Secure Group Communications Using Key Graphs. In *Proc. ACM SIGCOMM*, August 1998.