Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques

Isabella Chang Robert Engel
Dilip Kandlur Dimitrios Pendarakis Debanjan Saha

IBM T.J. Watson Research Center
{ichang,rengel,kandlur,dimitris,debanjan}@watson.ibm.com

Abstract—

The Internet today provides no support for privacy or authentication of multicast packets. However, an increasing number of applications will require secure multicast services in order to restrict group membership and enforce accountability of group members. A major problem associated with the deployment of secure multicast delivery services is the scalability of the key distribution protocol. This is particularly true with regard to the handling of group membership changes, such as member departures and/or expulsions, which necessitate the distribution of a new session key to all the remaining group members.

As the frequency of group membership changes increases, it becomes necessary to reduce the cost of key distribution operations. This paper explores the use of batching of group membership changes to reduce the frequency, and hence the cost, of key re-distribution operations. It focuses explicitly on the problem of *cumulative member removal* **and presents an algorithm that minimizes the number of messages required to distribute new keys to the remaining group members. The algorithm is used in conjunction with a new multicast key management scheme which uses a set of auxiliary keys in order to improve scalability. In contrast to previous schemes which generate a fixed hierarchy of keys, the proposed scheme dynamically generates the most suitable key hierarchy by composing different keys. Our cumulative member removal algorithm uses Boolean function minimization techniques, and outperforms all other schemes known to us in terms of message complexity.**

I. INTRODUCTION

The Internet today supports a basic form of multicast service. Receivers can join and leave a multicast group, identified by a Class D IP address, by sending IGMP messages to their local routers [8]. To send datagrams to a multicast group, a sender need not be a member of the group. It can simply address the datagrams to the group address. It is the responsibility of the multicast capable routers to communicate with each other using multicast routing protocols and deliver the datagrams to all members of the group. The multicast group is an open group and senders do not know the identities of the receivers in the group. Likewise, receivers do not have any mechanisms available to authenticate the identity of the senders.

Support for privacy and authentication in multicast distribution can be useful in a number of applications where it is important to restrict the set of receivers and/or authenticate the data source. Applications such as pay-per-view distribution of digital media, pay-per-use multi-party games, and restricted conferences fall in the category where the receiver set needs to be restricted to legitimate subscribers. In addition, for applications such as multicast distribution of stock market information it is also important to authenticate the data source.

To add secure services on top of IP multicast, each secure multicast group is usually associated with one or more trusted servers responsible for managing membership to the group. We refer to these servers as group controllers, or simply controllers, in this paper. In order to *join* a secure multicast session, either as a sender or as a receiver, a client has to request access to the group from the controller responsible for the session. Upon receiving a request from a client to join a secure multicast session, the controller examines the client credentials, in the form of a login name and password or a digital certificate. If the client is permitted to join the group, the controller provides it with the requisite keys as well as the multicast address where the client should listen for future control and data messages. The keys sent to the client include the session key which is shared by all members of the session and possibly, auxiliary keys, depending upon the key distribution algorithm.

The group controller is also responsible for handling client de-registration and removal. De-registration is initiated by a client and is important in applications such as pay-per-view where a client leaving a group would like to ensure that it is no longer charged for usage. Removal of a group member can be initiated by that member's domain controller and is important in cases where the member in question loses the access control privileges for the multicast group, due to lack of authentication, credit, etc.

Client de-registration and removal pose a complex scalability problem for multicast key management. To illustrate the problem, consider a secure multicast group consisting of n members, sharing a session key which is used for data transmissions. Now, assume one member of the group has to be removed. The session key has to be changed and communicated to *all* remaining $(n-1)$ members of the group. This will guarantee that the removed member cannot decrypt any future group communication and, furthermore, it cannot send any legitimate data to the group. However, communicating the new session key in a *scalable and secure fashion*, to the $(n-1)$ remaining members of the group is a non-trivial task.

0-7803-5420-6/99/$10.00 (c) 1999 IEEE

The simplest solution is to use a separate secure unicast connection from the controller to each remaining group member; this assumes that for each client there exists a unique key that is shared between the client and the controller. While simple, this solution suffers from poor scalability since it requires $(n-1)$ secure unicast connections and (n) secret keys.

The topic of key management for multiparty communication has been studied in the literature [10], [11], [3], [5]. However, with the exception of [4], [17] the scalability problem associated with frequent key changes in a large group has not been addressed. In Iolus [4] the scalability problem is addressed by dividing a large group into multiple subgroups and employing a hierarchy of group security agents. The scheme proposed in [17] uses a hierarchy of keys to solve the scalability problem. In this scheme, a key update requires $O(logN)$ messages where N is the size of the group. Each client has to maintain a key ring of $O(logN)$ keys and the controller has to manage a tree of $O(N)$ keys.

Our approach is similar to the scheme proposed in [17] in the sense that it uses a smart distribution of keys to achieve good scaling. However, instead of using a fixed hierarchy of keys, we dynamically generate the most suitable key hierarchy by composing different keys. As in [17], our scheme requires the controller to send $O(logN)$ messages to expel a single member from a group of size N. However, in multicast groups of large size and frequent membership changes, the modification and distribution of new session keys is an expensive operation, especially when it is used for each individual member departure. Instead, it is more likely that new keys will be generated at periodic intervals of time and/or in response to a significant number of member departures or expulsions.

This paper focuses explicitly on the problem of *cumulative member removal* and proposes a scheme that can be used to find the minimum number of messages required to distribute new keys to the remaining group members. Using Boolean function minimization techniques, our scheme outperforms all other schemes known to us in terms of message complexity in removing multiple group members simultaneously. A further advantage of our scheme is that the controller has to maintain only $O(logN)$ keys as opposed to $O(N)$ in [17]. A detailed comparison of our approach and the approach proposed in [17] is presented in section IV. Note that both our scheme and the one of [17] can be used in conjunction with Iolus [4].

The rest of the paper is organized as follows. In section II we present our key management and distribution scheme. Section III contains an analysis of the proposed scheme. A comparison of our approach with other schemes proposed in the literature can be found in section IV. Finally, in section V we draw our conclusions.

II. Key Management Scheme

In our scheme each member of the group is associated with a unique user ID (UID) which is a binary string of length n. Consequently, a UID can be written as $X_{n-1}X_{n-2}\ldots X_0$, where X_i can be either 0 or 1. Using boolean notation, X_i can be written as $\bar{x}_i$ or x_i depending on whether X_i is 0 or 1. The length of the UID depends upon the size of the multicast group. For example, in a group with more than 4 and up to 8 members we can use 3 bit UIDs.

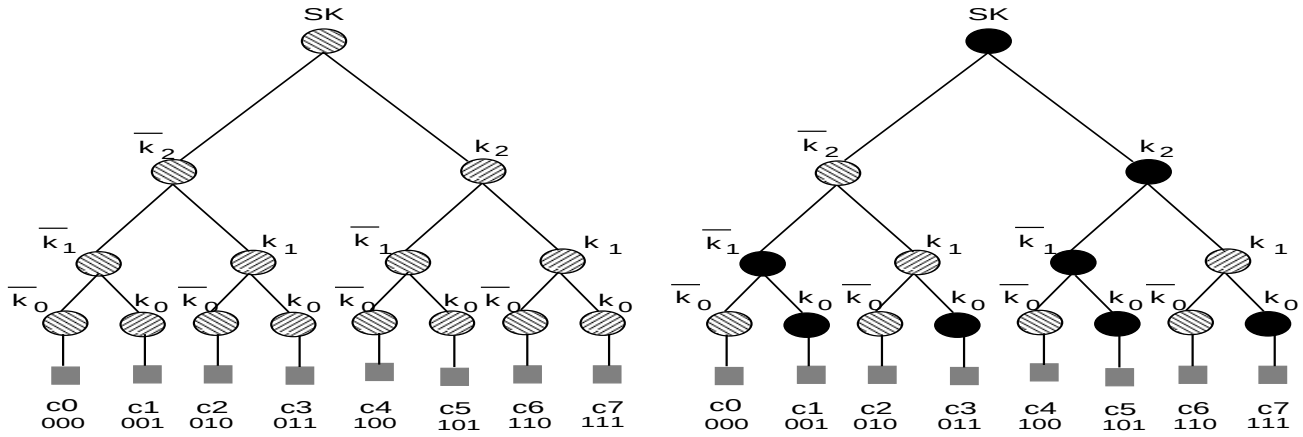
When a member with UID $X_{n-1}X_{n-2}\ldots X_0$ registers with the group controller to join a session it receives the common session key, SK. The session key is shared by all current members of the group and is used to encrypt/decrypt data messages sent to the multicast group. Additionally, the member receives a set of n *auxiliary keys* $K_{n-1}, K_{n-2}, \ldots, K_0$, where K_i is written as k_i if $X_i = 1$ and $\bar{k}_i$ if $X_i = 0$ [1]. The auxiliary keys are used to update session keys in a secure manner. These keys are drawn from a set of n key pairs. Each key pair corresponds to a bit in the UID. Each member receives exactly one key out of every key pair. The controller manages all the auxiliary keys, namely $\{k_0, \bar{k}_0, k_1, \bar{k}_1, \ldots, k_{n-1}, \bar{k}_{n-1}\}$. Figure 1(a) shows an example of keys possessed by different members in a group of size 8. The square leaf nodes in the tree represent the members in the group. As shown in the figure, each member is identified by a unique 3-bit UID. The round nodes in the tree represent the keys in the system. Notice that there are three levels in the tree, each corresponding to a bit position in the UID. Each member possesses all the keys on the branch from the leaf representing its UID to the root of the tree. For example, member c_5 (UID 101) possesses the auxiliary keys k_2,$\bar{k}_1$, and k_0 in addition to the session key SK which is shared by all members.

In general, both the session and the auxiliary keys change whenever a group member de-registers or is to be removed, so that it can no longer send to or receive messages addressed to the group. We term the event of such change of keys as *group re-keying*. Since group re-keying takes place in discrete times, we denote the session key as $SK(r)$ and the auxiliary keys as $k_i(r)$ or $\bar{k}_i(r)$, where r is the *current round number*. We define a round to be the sequence number, starting from 0 at the time the multicast session is created, of an interval in which the session key remains unchanged.

A. Individual Member Removal

Whenever a member of a multicast group is to be expelled, e.g., because its subscription has expired, a new session key needs to be distributed to every member except the one leaving to make sure that the expelled member can no longer receive and send data addressed to the group. Similarly, if a member voluntarily leaves the multicast group, the session key might also have to be updated, depending on the re-keying policy of the group controller. This can be useful for sessions where members pay according to the duration of their membership in the group.

[1] Note that values k_i and $\bar{k}_i$ are not complements of each other, they are two unrelated keys.



(a) Key distribution (b) Departure of c_5

Fig. 1. Key distribution and update in a group of size 8.

In order to update session key $SK(r)$, the controller computes a new session key $SK(r+1)$. The new session key is encrypted with the keys that are "complementary" to the ones of the departing member. For example, assume that the departing member's UID is 101. Therefore, it possesses keys $k_2, \bar{k}_1, k_0$. The new session key is encrypted and sent in 3 different messages, $\{SK(r+1)\}_{\bar{k}_0}, \{SK(r+1)\}_{k_1}, \{SK(r+1)\}_{\bar{k}_2}$, where $\{L\}_M$ means that the string L is encrypted with key M, and is multicast to the entire group. Although the departing member receives all the messages, it can not decrypt them, since every message is encrypted with a key that the departing member does not possess. It is also guaranteed that every other member of the group can decrypt at least one message. This is due to the fact that the UID of every other member differs from the UID of the departing member in at least one bit position, and therefore their key sets differ as well in at least one key. This differing key(s) can be used for decryption.

Figure 1(b) shows a visual interpretation of the re-keying scheme described above using the example in Figure 1(a). In the figure, the keys corresponding to the solid round nodes correspond to the keys possessed by the departing member c_5. The hatched round nodes represent the complementary set, that is, the keys not possessed by c_5. Note that the branch from c_5 to the root of the tree has only solid round nodes. Every other branch has at least one hatched node. Hence, if the new session is encrypted individually with the keys not possessed by member c_5, all members except for c_5 will be able to decrypt at least one of the messages.

A simple analysis of this key distribution algorithm shows that for a group of N members the number of keys that need to be maintained by the controller is of the order $O(logN)$ and that the number of messages that need to be sent out to update the session key after the removal of a single member is $O(logN)$, with each message encrypted with one key. Alternatively, we can pack the three encrypted keys in one single message. Packing multiple encrypted payloads in a single message reduces the overhead if the message has to be signed by the controller to ensure authenticity.

Clearly, in this scheme the departing member is excluded from learning the new session key. To make sure that it can not use its auxiliary keys to decrypt future session key updates, auxiliary keys are updated as well. To update key $K_i(r)$, a one way hash function f is used that yields the updated auxiliary key as follows $K_i(r+1) = f(K_i(r), SK(r+1))$. This guarantees that only a member that is in possession of the new session key $SK(r+1)$ can obtain the updated auxiliary key $K_i(r+1)$. Since the departing member does not know the new session key $SK(r+1)$, it is excluded form the future updates of the session key.

B. Removal of Multiple Members

The key update procedure described in the previous section can be applied k times successively to remove k members from the multicast group. However, a more efficient strategy is to aggregate the removal of several members from the group. This can be useful for policies where key updating is done only in certain intervals to save resources, or when several members are expelled/depart either simultaneously or within a very small time interval.

In this section we present a systematic approach to the problem of removing members in the same round. In general, consider a set of clients, $S = \{c_0, c_1, \ldots, c_{N-1}\}$, where $N = 2^n$. The user ID (UID) of a client c is written, in binary form, as an n-bit ID $u(c) = X_{n-1}X_{n-2}\ldots X_0$, where X_i, $i = 0, 1, \ldots, n-1$ is either 0 or 1. At any point in time, membership in the secure multicast group can be determined by a Boolean function $m()$ of the UID. That is, if $m(X_0, X_1, \ldots, X_{n-1}) = 1$, the client with UID $X_{n-1}X_{n-2}\ldots X_0$ is in the group. Otherwise, that client is to be excluded from the group.

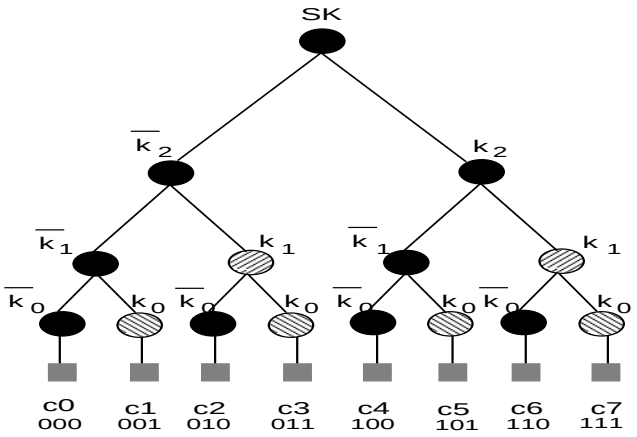


Fig. 2. Example of multiple members departing in the same round.

Group re-keying can be achieved by updating the session and auxiliary keys for all but the clients for which $m(u) = 0$. To this extent, re-keying information has to be encrypted with keys unknown to the excluded clients. For scalability and efficiency reasons, it is desired that re-keying requires a minimum number of messages sent to the group and/or that a minimum number of encryption operations are performed. Intuitively, this is achieved by encrypting re-keying information with keys common to subsets of the remaining members.

Consider the example in Figure 1. Suppose that members c_0 with UID 000 and c_4 with UID 100 have to be removed from the group. The objective of re-keying is to provide the new session key, denoted by $SK(l+1)$, to the remaining members, $S_r = \{c_1, c_2, c_3, c_5, c_6, c_7\}$. Note that, in general, the set of the actual remaining group members is a subset of S_r since not all UIDs may be assigned. UIDs that are not assigned can be treated as "don't care" conditions without any impact in the procedure. In this example, it is sufficient to multicast the following two messages containing the encrypted new session key, $\{SK(l+1)\}_{k_0}$ and $\{SK(l+1)\}_{k_1}$. The first message can be decrypted only by c_1, c_3, c_5, c_7 and the second only by members c_2, c_3, c_6, c_7. Hence, the combination of these two messages covers S_r and is thus sufficient for group re-keying.

Figure 2 shows a visual interpretation of the re-keying scheme described above. The solid round nodes represent the keys possessed by either c_0 or c_4. Clearly, these keys may not be used to encrypt the new session key. However, as mentioned above, the keys k_0 and k_1 cover the remaining set of members. Hence, two messages, each containing the new session key encrypted individually with k_0 and k_1 is sufficient to update the keys for the rest of the group.

Observe that instead of sending out a total of $2 * 3 = 6$ messages, as would have been required if re-keying were performed sequentially, we only need 2 messages by aggregating the removal of both members. This number is in fact one less than the number of messages required to remove a single member. Intuitively, the number of messages is reduced when the UIDs of the remaining group members, have one or several bits in common that are different from the excluded members' UIDs. In the above example, members c_2, c_3, c_6, c_7 all have the same value in bit X_1 of their UID which is different from the value of X_1 for the excluded members. This translates into a key, k_1 which is known to these members but not to the excluded ones (they possess $\bar{k}_1$). Similarly, members c_1, c_3, c_5, c_7 have the same value for bit X_0 and all possess k_0 while the excluded members possess $\bar{k}_0$ and can not decrypt the message.

Thus, the problem of cumulative group member removal becomes one of grouping the remaining members that share common bits in their UIDs (and hence common keys) which are different from those of the removed members, in an efficient and systematic way. Formally, this problem is equivalent to the *minimization* of the Boolean membership function $m()$. For the example presented above, the membership function can be written as:

$$\begin{aligned} m(X_2, X_1, X_0) \quad = \quad & \bar{X}_2\bar{X}_1X_0 + \bar{X}_2X_1\bar{X}_0 + \bar{X}_2X_1X_0 \\ & + X_2\bar{X}_1X_0 + X_2X_1\bar{X}_0 + X_2X_1X_0 \ , \end{aligned}$$

where $+$ represents logical OR and multiplication represents logical AND. In other words, the membership function evaluates to 1 for the UIDs of all members of the group. The form of equation 1 suggests a straightforward solution for re-keying the group: multicast 6 messages, one for each term of the sum, where each message is encrypted with a key which is a function of the keys corresponding to that term. Such a function could be a one way hash function applied to all the elementary keys, that yields the composite key with which the message is encrypted. For example, term $\bar{X}_2\bar{X}_1X_0$ corresponds to a message encrypted with a key derived from keys $\bar{k}_2, \bar{k}_1, k_0$. It is obvious that each one of these messages can be decrypted by one and only one of the remaining group members; there is no aggregation and no utilization of keys which might be common among different remaining members.

Therefore, the need for simplification and aggregation arises. Similar problems have been addressed for many years in the area of switching theory and logical design. The objective there is to minimize Boolean functions so that the complexity of digital circuits can be reduced. In the context of logical design, a $+$ operation corresponds to an OR gate and a multiplication to an AND gate. Typical objectives include the minimization of total number of gates and/or number of circuit stages.

We borrow from the results of logical design to construct a more efficient re-keying process. First, we define some of the terms we use in subsequent discussions.

- *Literal*: A variable or its complement, e.g., x_1, $\bar{x}_1$, x_2, $\bar{x}_2$, etc.
- *Product Term*: Series of literals related by AND, e.g., $\bar{x}_1\bar{x}_2x_3$, $\bar{x}_1x_2\bar{x}_3$, etc.

- *Minterm*: A product term which contains as many literals as there are variables in the function. In equation 1, all products are minterms.
- *Sum term*: Series of literals related by OR, e.g., $x_1 + x_3$, $\bar{x}_2 + x_3$.
- *Normal term*: Product or sum term in which no variable appears more than once.

The standard form most usually considered in the simplification of Boolean functions is the form known as the *sum of products expression (SOPE)*. In the context of logical design, each product corresponds to an AND gate and each literal to a gate input. In the context of the multicast group re-keying problem, each product corresponds to a message and each literal to a key which is used as input to a function that derives the encryption/decryption key for the message. Many criteria can be applied in optimizing a sum of products form. In the context of logical design, a sum of products expression is regarded as a *minimal* expression if there exists (1) no other equivalent expression involving fewer products, and (2) no other expression involving the same number of products but a smaller number of literals.

The rationale behind this definition of optimality is that typically the cost of an additional gate is several times that of an additional input on an already existing gate and, hence, elimination of gates is the primary objective of the minimization process. Interestingly, the same definition of optimality is also applicable to our problem. The argument in our case is that the complexity of sending an additional message is far greater that of adding an extra key ID in the message to indicate that the key should be used as input in deriving a new key.

In deriving a minimal expression, the Karnaugh map [2] representation of boolean functions can be used. Karnaugh maps provide an intuitive visual technique that helps to identify product terms. However, for large number of variables this method becomes hard to use since it is essentially a trial-and-error method that relies on the ability to recognize patterns. A systematic approach applicable to complex functions was developed by Quine and improved by McCluskey. It provides a step by step approach on how to find out the minimum number of SOPE, using a tabular method that can easily be implemented on a computer. More details can be found in [6]. The controller executes the Quine-McCluskey algorithm to compute the messages that need to be sent out after multiple members depart the group in the same round.

To understand how the boolean minimization applies to our problem of key updates with a minimum number of messages, consider the again the example in Figure 1. Assume that we are currently in a state where the group has 7 members and all UIDs are assigned but 101 (c_5). Suppose, we now have to remove c_0 and c_4 from the group. Table I shows membership function $m()$ for the group after the departure of c_0 and c_4. Note that the output for UIDs corresponding to c_0 and c_4 is 0. The output corresponding to all other members, except for c_5 is 1. Since the UID corresponding to c_5 is not assigned in this round, the output is set to "don't care"and is indicated by X. Intuitively, since the auxiliary keys were updated after c_5 left the group, it would not be able to decrypt the messages encrypted with the new keys.

Input $(X_2 X_1 X_0)$	Output
000	0
001	1
010	1
011	1
100	0
101	X
110	1
111	1

TABLE I
BOOLEAN MEMBERSHIP FUNCTION.

Figure 3(a) shows the Karnaugh map representation of the membership function. Each field of the Karnaugh map corresponds to a specific minterm and is marked 0, 1 or X. The next step in the minimization procedure is to identify the largest possible "rectangles" that contain only 1 and X as shown in Figure 3(b). These rectangles are called prime implicants of the function. By choosing the minimum number of prime implicants the minimum SOPE of the function is obtained. For this example, the function can be minimized to $x_1 + x_0$.

There is a straightforward analogy between minimizing boolean functions and aggregating re-keying messages. The interpretation of Figure 3 is as follows.

- The fields containing a 0 correspond to the members that have to be removed from the group.
- The fields containing a 1 correspond to the remaining members in the group that need to be updated with the new session key.
- The fields containing an X correspond to UIDs that have not been assigned yet.

Therefore, updating session and auxiliary keys after cumulative removal of a set of group members reduces to finding the minimum number of blocks in the Karnaugh table, so that all 1s are in a block, but none of the 0s. Each block can then be mapped into a message that is encrypted with the identifier of the block and multicast out to the group.

Due to the minimal number of auxiliary keys that our key management and distribution scheme maintains, it may be susceptible to collusion attacks. In a collusion attack, a set of members previously removed from the group collude and by combining their sets of keys may be able to obtain the currently valid set of keys, thereby being able to continue unauthorized receipt of group communication. To illustrate this

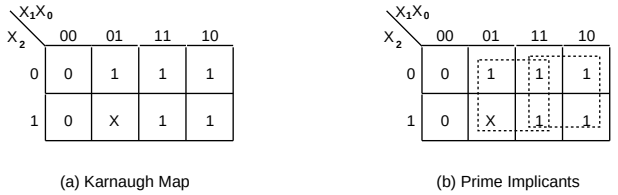


Fig. 3. Karnaugh map minimization of membership function.

problem, consider a group of 8 members in which two members, with IDs 000 and 111 respectively, are to be removed in the same round. As can be easily observed, these members can mount a collusion attack since they collectively hold all auxiliary keys for this group. While it is impossible to eliminate the risk of a collusion attack with less than $O(n)$ auxiliary keys, it is possible to raise the bar with a larger auxiliary key space and a sparse distribution of UIDs.

III. Properties and Performance

In this section we analyze the performance of the proposed key management scheme. We focus on the performance in terms of the total number of messages required to update the session key when multiple members leave the group. In section III-A, we derive the worst case complexity for two cases: first the departure of *two* members of a secure multicast group and then the departure of $N/2 = 2^{n-1}$ members. Then, in section III-B we present an upper bound on the *average* number of messages required for aggregate removal of an arbitrary number of members.

A. Worst Case Performance

A.1 Cumulative Removal of 2 Group Members

We examine the case when new keys have to be distributed after 2 clients, C_1 and C_2, are removed from a secure multicast group. Assume the client IDs, in binary representation, are $X_{n-1}X_{n-2}\ldots X_0$ and $X'_{n-1}X'_{n-2}\ldots X'_0$, respectively. Since clients are represented by an n-bit ID, there can be a total of 2^n of them. Without loss of generality we assume that $X_{n-1}X_{n-2}\ldots X_0 = x_{n-1}x_{n-2}\ldots x_0$; the discussion in this section holds for any value of the UID of C_1.

We first assume that the Hamming distance between the two UIDs is maximum, i.e., that

$$X'_i = \bar{X}_i = \bar{x}_i \ , i = 0, 1, \ldots, n-1,$$

where $\bar{x}$ denotes the 2's complement of x. Hence, client C_1 posseses keys $k_{n-1}, k_{n-2}, \ldots, k_1, k_0$ and client C_2 keys $\bar{k}_{n-1}, \bar{k}_{n-2}, \ldots, \bar{k}_1, \bar{k}_0$. As we will prove next, this is the worst case in terms of messages that have to be sent to update the remaining members.

We will show that at most n messages are needed for group re-keying. First, we claim that the following n messages can be used to distribute the new keying information to all the remaining group members.

$$\{SK(r+1)\}_{f(k_{n-1},\bar{k}_{n-2})}, \quad \{SK(r+1)\}_{f(k_{n-2},\bar{k}_{n-3})}, \quad \ldots, \quad \{SK(r+1)\}_{f(k_0,\bar{k}_{n-1})}$$

Each of the n terms in the above expression corresponds to a single message which contains the new session key $SK(r+1)$ encrypted with a composite key. The composite key used for this encryption is derived from a one-way function $f(.,.)$ of two keys. One straightforward option is to assume $f(K_1, K_2) = K_1K_2$ i.e., $SK(r+1)$ is doubly encrypted with both keys. Alternatively, one can employ a one-way function, like a hash function, which computes a single key from its two arguments, thus avoiding the extra cost of double encryption.

We now show that the n messages of expression 1 are sufficient for group re-keying. First, observe that neither C_1 nor C_2 are able to decrypt any of these messages, since each one of them possesses one and only one of the two keys used to encrypt each message. Next, we will show that every one of the remaining members of the multicast group can decrypt at least one of the above n messages.

Lemma 1: Excluding C_1 and C_2, every member of the secure multicast group can decrypt at least one of the n messages of expression 1.

Proof. Consider an arbitrary remaining member of the group, C, with UID $y_{n-1}y_{n-2}\ldots y_1$, which is obviously different from the IDs of C_1 and C_2. Let m be the highest order bit in which the IDs of C and C_1 differ (there has to be such a bit otherwise $C \equiv C_1$), i.e.,

$$y_i = x_i, \ \ i = n-1, n-2, \ldots, m+1$$

$$y_m = \bar{x}_m.$$

If $m < n-1$, C possesses both k_{m+1} and $\bar{k}_m$ and, therefore, can decrypt the $(n-(m+1))$th message in expression 1.

If $m = n-1$, i.e., $y_{n-1} = \bar{x}_{n-1}$, let l be the lowest order bit in which the IDs of C and C_1 match (there has to be such a bit, otherwise $C \equiv C_2$), i.e.,

$$y_l = x_l, \ \ 0 \leq l < n-1$$

$$y_i = \bar{x}_i, \ \ i = l-1, l-2, \ldots, 0.$$

If $l = 0$, C possesses both k_0 and $\bar{k}_{n-1}$, so it can decrypt the nth message in expression 1. Otherwise, C possesses both k_l and $\bar{k}_{l-1}$, so it can decrypt the $(n-l)$th message in expression 1.

Theorem 1: Re-keying a secure multicast group of size 2^n when two group members are to be removed requires at most n messages.

Proof. If the IDs of the two users, denoted by C_1 and C_2, differ in all bits (i.e., have maximum Hamming distance), Lemma 1 applies.

Otherwise, the IDs have at least one bit in common; let this be X_i. Observe that a message encrypted with the key that corresponds to the complement of X_i, i.e., k_i if $X_i = 0$ or $\bar{k}_i$ if $X_i = 1$, is sufficient to distribute the new keying information to half (2^{n-1}) of the group members, while excluding C_1 and C_2. The remaining $2^{n-1} - 2$ members which also belong to the group, together with C_1 and C_2, all have X_i as the i-th bit in their IDs. Hence, this bit can be effectively ignored and thus the problem reduces itself to that of removing 2 users from a group of 2^{n-1} members, whose IDs have $(n-1)$ bits.

This procedure can be applied recursively, yielding one message for every common bit in C_1 and C_2. After the i-th message, 2^{n-i} users are left, including C_1 and C_2. Therefore, for k common bits in the UIDs of C_1 and C_2, the solution is comprised of two steps.

- Generate k messages which convey the new keying information to $2^n - 2^{n-k}$ clients.
- Re-key a group of 2^{n-k} clients whose IDs are of size $(n-k)$ bits and where the Hamming distance between the IDs of the two users removed is maximum. As shown in Lemma 1 this problem is solved with $(n-k)$ messages.

Summing over the two steps, this solution requires n messages. If we assume, without loss of generality that the common bits are the k first, as follows,

$$C_1 = x_{n-1}x_{n-2}\ldots x_{(n-k)}x_{(n-(k+1))}\ldots x_0 \ ,$$

$$C_2 = x_{n-1}x_{n-2}\ldots x_{(n-k)}\bar{x}_{(n-(k+1))}\ldots \bar{x}_0 \ ,$$

then these messages can be expressed as follows:

$$\begin{array}{rr} \{SK(r+1)\}_{f(\bar{k}_{n-1})}, & \{SK(r+1)\}_{f(\bar{k}_{n-2})}, \\ \{SK(r+1)\}_{f(\bar{k}_{n-k})}, & \{SK(r+1)\}_{f(k_{(n-(k+1))},\bar{k}_{(n-(k+2))})}, \\ \ldots, & \{SK(r+1)\}_{f(k_0,\bar{k}_{(n-(k+1))})} \ . \end{array}$$

Note that if $k = n-1$, the k first messages are sufficient, as can be easily verified.

A.2 Aggregate Removal of 2^{n-1} Members

Consider a secure multicast group of $N = 2^n$ members where new keys have to be distributed after $N/2 = 2^{n-1}$ clients are removed from the group. We claim that the worst case complexity corresponds to the case where the UIDs of the $N/2$ remaining members are assigned such that the Hamming distance between any two UIDs is 2 or larger. This means that for any UID $X_{n-1}X_{n-2}\ldots X_0$ of a remaining member, all n UIDs which differ in only one bit were assigned to departing members, or any remaining member is "encircled" by departing members, as shown in the Karnaugh table of Figure 4. Intuitively, this means that no remaining member can be grouped with another remaining member in the optimization of the Boolean membership function. As a result, re-keying requires one message to be sent out for every remaining group members, a total of $N/2$ messages.

X_3X_2 \ X_1X_0	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

Fig. 4. Worst case for aggregate removal of $2^{(n-1)}$ members.

Now we will prove that $N/2$ is the absolute worst case number in terms of update messages that need to be sent out for any number of departing members.

To show this, first consider the case when the number of departing members is greater or equal to $N/2$ and, thus, the number of remaining members is $N/2$ or less. Clearly, the number of messages required is no more than the number of remaining members, since at worst one has to send out one message to update a remaining member or one message per minterm of the membership function. Hence, in this case the number of messages required will be at most $N/2$.

Now consider the case when the number of departing members is less than $N/2$ and, thus, the number of remaining members is greater than $N/2$. Looking at Figure 4, for every additional member which remains in the group or, equivalently, for each additional minterm of the membership function, there exists (at least) one previously existing minterm of the function with which it can be grouped. Grouping can be done in a systematic way so that every additional member is grouped with one and only one of the existing ones. For example by pairing with the member with the largest UID smaller than that of itself. This argument holds even if the existing $N/2$ minterms are not placed as shown in Figure 4, since moving any of the minterms shown to any other position in the table will reduce the number of messages by 1.

B. Average Case Performance

It is interesting to study the average number of messages required for the aggregate removal of an *arbitrary* number of members from a multicast group.

In the logic minimization literature, the *average number of products* in the *minimum sum-of-products expressions (SOPE)* of Boolean functions has been studied extensively. As mentioned earlier, for switching functions, the average number of products in SOPE's is equal to the average number of AND gates in minimum AND-OR two level circuits, while in our context it is equal to the average number of messages required

for re-keying.

Sasao in [1] derives an upper bound on the average number of products in the minimum SOPE's that are claimed to be the tightest reported to date. The bound considers p-valued input two-valued output functions, a more general case than the case of $p = 2$ which has been considered throughout this paper.

For reasons of completeness, we now present the result, adapted from [1]. We begin by introducing the concept of VP-equivalence relation and proceed with the theorem stating the upper bound.

Definition 1: The relation $\sim$ satisfying the following conditions, where f is a n-variable p-valued input two valued output function, is called VP-equivalence relation.

1. $f \sim f$.
2. If $f_1 = f(\ldots, X_i, \ldots, X_j, \ldots)$ and $f_2 = f(\ldots, X_j, \ldots, X_i, \ldots)$, then $f_1 \sim f_2$ (permutation of input variables).
3. Let $\sigma : P \rightarrow P$ be an arbitrary one-to-one mapping, where $P = \{0, 1, \ldots, p-1\}$ is the set of truth values that $X_i,\ i = 1, 2, \ldots, n$ take values in. If $f_1 = (\ldots, X_i, \ldots)$ and $f_2 = (\ldots, \sigma(X_i), \ldots)$, then $f_1 \sim f_2$ (permutation of values in a variable).

For $p = 2$, VP-equivalence is called NP-equivalence.

Theorem 2: Let $U_p(n, u)$ be an upper bound on the average number of products in minimum SOPE's for n-variable p-valued input two-valued output functions for a given u, the number of elements in $f^{-1}(1)$ i.e., the number of minterms of f (u is also called the weight of f and denoted by $|f|$). Then

$$U_p(n, u) = \frac{p^{n-k}}{F^{(u)}} \sum_{j=1}^{p^k} c(j) \cdot \begin{pmatrix} w - p^k \\ u - j \end{pmatrix} , \qquad (1)$$

where

$$c(j) = \sum_{|g_i|=j} \mu(g_i) \cdot t(g_i) ,$$

and $g_1, g_2, \ldots g_\lambda$ are representative functions of VP-equivalence classes, $t(g_i)$ is the number of products in a minimum SOPE for g_i, and $\mu(g_i)$ is the number of functions which are VP-equivalent to g_i.

In equation 1, $w = p^n$ is the number of all possible different n-variable p-valued input two-valued output functions and $F^{(u)} = \begin{pmatrix} w \\ u \end{pmatrix}$ is the number of different functions with weight u. The upper bound is a function of the variable $k \leq n$. In general, the larger the k, the tighter the upper bound, but the larger the computational complexity since the number of VP equivalence classes. [1] contains representative examples of the $c(j)$ coefficients.

Proof. See [1].

IV. Related Work

Several authors have addressed the problem of providing security in multiparty communication. For an overview of the various issues related to multicast security please refer to [13] and [14]. Among the works closely related to the work presented here, GKMP [10], [11] is one of the most prominent. In GKMP a group member is selected to be the *Group Key Controller* responsible for handling join requests and distributing (new) keys. The Group Key Controller generates the group keys in a joint operation with a selected group member. It then contacts each valid group member and sends it the group keys encrypted by a key mutually shared between the Group Key Controller and that member. This approach suffers from scalability problems since a single entity is involved in sending the group keys to every member on a one by one basis, encrypted with the specific shared key

Another prominent multicast key management protocol SMKD [3] works in conjuction with the Core Based Tree (CBT) [12] multicast protocol. It allows members to securely join a CBT group tree. SMKD enhances the scalability by exploiting the implicit hierarchy of the CBT distribution tree and the fact that routers on the delivery tree know the identities of their tree neighbors. When a CBT tree is first constructed, the tree root operates as group controller responsible for group key generation and distribution. The ability to distribute the group keys further is delegated to other routers as they join the delivery tree. SKMD achieves a high scalability but it does not offer a satisfactory solution to the re-keying problem in the case of frequent group membership changes. Furthermore, it is vulnerable to breach of security by "corrupt" routers in the distribution tree.

In Iolus [4] the scalability problem is addressed by dividing a secure multicast group into multiple sub-groups organized in a multi-level hierarchy. The main focus of that work is the architecture of the group hierarchy and inter-group key management. In Iolus terminology, our work focuses more on the key management within a single "subgroup", a topic not addressed in [4].

In terms of addressing the scalability problem of group key management, the scheme proposed in [17] is the one closest to ours. In this scheme, clients are organized in a virtual hierarchy as shown in figure 5(a). Each round node in the tree represents a key with the label being the key ID. The label for the root of each tree represents the session key of the group. A client, represented by a square leaf node, possesses all the keys on the branch from the leaf to the root of the tree. When a client leaves or gets expelled from the group, all keys on the branch from the leaf representing the client to the root are compromised and have to be changed. However, updated keys can now be multicast to the sub-groups instead of being unicast to individual members of the group. For example, when member c_5 leaves the group, keys K_{45},K_{4567}, and $K_{01234567}$ are updated. Key K_{45} is encrypted with key K_4, key K_{4567} is

encrypted with key K_{67} and new key K_{45}, and key $K_{01234567}$ is encrypted with key K_{0123} and new key K_{4567} and multicast to the entire group. In this scheme, a key update requires $O(logN)$ messages where N is the size of the group. Note that each client has to manage $O(logN)$ keys and the controller has to manage a tree of $O(N)$ keys. This scheme is similar to our scheme in terms of message complexity for individual member removal.

However, the number of different keys maintained by the controller is $O(n)$ compared to $O(2log(n))$ for our approach. Since generating new keys can be expensive, this reduction in the number of keys can be a significant advantage of our approach. Keeping track of key assignment is also easier in our case since there is a functional mapping of UIDs to keys. Most importantly, our scheme is superior in minimizing the number of messages when multiple members leave the session in the same round. In the following, we explain this in more detail.

Figure 5(b), shows the distribution of keys among the members in a group of 8 in our scheme. We can show that the key hierarchy in Figure 5(a) is a special case of a number of key hierarchies that can be dynamically generated in our scheme. This can be accomplished by defining a mapping between the two hierarchies that can be used to compute the key hierarchy with 15 different composite keys of figure 5(a) from the set of 6 elementary keys and SK of figure 5(b). In the mapping $R : C \rightarrow E$ defined below, E is the set of keys in figure 5(b), C is the set of keys in figure 5(a), and f is an one-way hash function.

$$
\begin{aligned}
E &= \{k_2; \bar{k}_2; k_1; \bar{k}_1; k_0; \bar{k}_0; SK\} \\
C &= \{K_0; K_1; K_2; K_3; K_4; K_5; K_6; K_7; K_{01}; K_{23}; \\
&\quad K_{45}; K_{67}; K_{0123}; K_{4567}; K_{01234567}\} \\
R &= \{K_0 \rightarrow f(\bar{k}_2, \bar{k}_1, \bar{k}_0); K_1 \rightarrow f(\bar{k}_2, \bar{k}_1, k_0); \\
&\quad K_2 \rightarrow f(\bar{k}_2, k_1, \bar{k}_0); K_3 \rightarrow f(\bar{k}_2, k_1, k_0); \\
&\quad K_4 \rightarrow f(k_2, \bar{k}_1, \bar{k}_0); K_5 \rightarrow f(k_2, \bar{k}_1, k_0); \\
&\quad K_6 \rightarrow f(k_2, k_1, \bar{k}_0); K_7 \rightarrow f(k_2, k_1, k_0); \\
&\quad K_{01} \rightarrow f(\bar{k}_2, \bar{k}_1); K_{23} \rightarrow f(\bar{k}_2, k_1); \\
&\quad K_{45} \rightarrow f(k_2, \bar{k}_1); K_{67} \rightarrow f(k_2, k_1); \\
&\quad K_{0123} \rightarrow \bar{k}_2; K_{4567} \rightarrow k_2; K_{01234567} \rightarrow SK\}
\end{aligned}
$$

Clearly, this mapping can easily be extended for larger groups and key hierarchies. Note, that there are many different mappings from E to C. For example, we can swap any key pair $k_i/\bar{k}_i$ with $k_j/\bar{k}_j$ or k_i with $\bar{k}_i$. Or, we can easily redraw the key graph from Figure 5(b) by swapping the keys on the different levels without affecting the keys the members possess, while the composite key hierarchy has a fixed hierarchy of keys. These additional degrees of freedom translate into an increased number of key hierarchies that can be formed. This flexibility can be exploited in reducing the number of messages that need to be sent out to expel/remove multiple members in the same round.

For example, to remove members c_0, c_3, and c_5, using the scheme proposed in [17], [5] the controller needs to send 9 messages ($\{K_{01}\}_{K_1}$, $\{K_{23}\}_{K_2}$, $\{K_{45}\}_{K_4}$, $\{K_{0123}\}_{K_{01}}$, $\{K_{0123}\}_{K_{23}}$, $\{K_{4567}\}_{K_{45}}$, $\{K_{4567}\}_{K_{67}}$ $\{K_{01234567}\}_{K_{0123}}$, $\{K_{01234567}\}_{K_{4567}}$, where the controller always uses the new, updated key to encrypt). To remove the same members in our scheme it is enough to send out 4 messages ($\{SK\}_{k_2\bar{k}_0}$, $\{SK\}_{\bar{k}_2\bar{k}_1k_0}$, $\{SK\}_{k_2k_1}$, $\{SK\}_{k_1\bar{k}_0}$) that can be computed using the Karnaugh map.

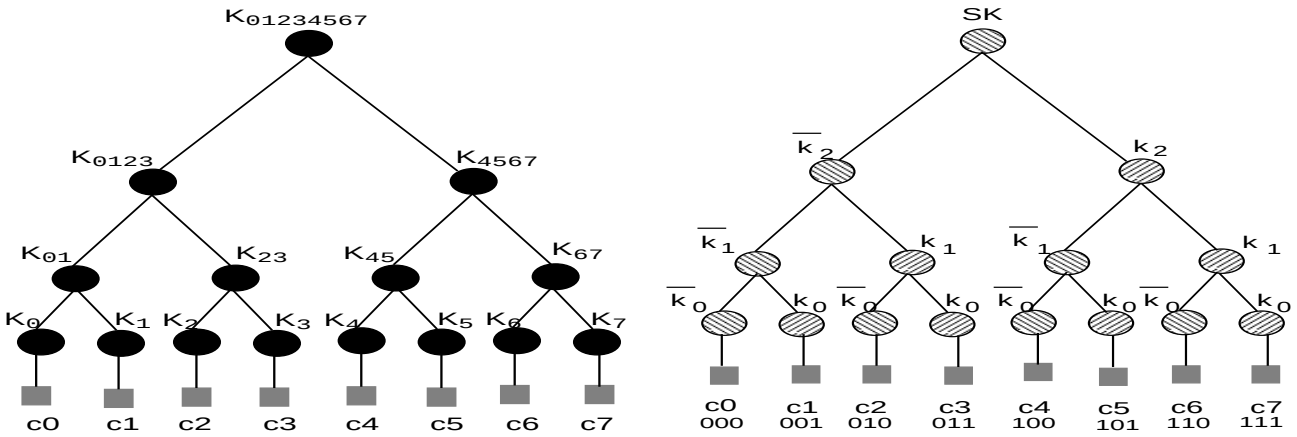
V. Conclusion

In this paper, we presented an efficient key management and distribution scheme for secure multicast, that scales extremely well in terms of group size and dynamics. For a multicast group of n members the number of keys that need to be maintained by the group controller is $O(log(n))$ and the message complexity associated with updating keys when a single member departs the group is also $O(log(n)$. When removal of multiple group members in the same round is desired, we use boolean minimization in order to find the minimum number of messages needed to update keys . In terms of message complexity in removing multiple members in the same round, our scheme outperforms all other schemes known to us. In fact, depending on the identity of the specific members leaving the group, the number of messages required to update keys is at times less than the number of messages required to expel a single member. In many secure multicast applications, such as pay-per-view events, members join and leave the group in bursts. In these applications, efficient removal of multiple members in the same round is critically important. Even in applications such as multi-party games, where membership changes are spread over time, aggregation of multiple departures into a single key update event is important for performance reasons. The efficiency of our scheme in aggregating key updates due to multiple departures can translate into a tremendous performance advantage.

This work can be extended in many ways. We are in the process of analyzing the average case overhead of the algorithm using simulations. The proposed scheme is being used within a toolkit for secure Internet multicast services that we have developed. We continue to benchmark and optimize various components of the toolkit, including the key distribution algorithm. We plan to use the toolkit to enhance a number of multicast applications with security, such as synchronization of web caches and proxies, updating of distributed databases and multi-party games.

VI. Acknowledgment

We would like to thank Tsutomu Sasao from the Kyushu Institute of Technology for his discussions on minimizing sum of product expressions (SOPE).



(a) Key hierarchy in [17] (b) Key hierarchy in our scheme

Fig. 5. Comparison of key hierarchies in our scheme and in the scheme proposed in [17].

REFERENCES

[1] Tsutomu Sasao, "Bounds on the Average Number of Products in the Minimum Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functions", *IEEE Transactions on Computers*, Vol. 40, No. 5, pp. 645–651, May 1991.

[2] M. Karnaugh, "The Map Method for Synthesis of Combinational Logic Circuits", *Transactions AIEE, Communications and Electronics*, Vol. 72, pp. 593-599, November 1953.

[3] A. Ballardie, "Scalable Multicast Key Distribution" RFC 1949, May 1996.

[4] Suvo Mittra, "Iolus: A Framework for Scalable Secure Multicasting", *Proceedings of ACM SIGCOMM'97*, Cannes, France, pp. 277–288, 1997.

[5] Debby M. Wallner, Eric J. Harder, Ryan C. Agee, "Key Management for Multicast: Issues and Architectures", *Informational RFC, draft-wallner-key-arch-00.txt*, July 1997.

[6] E. J. McCluskey Jr., "Minimization of Boolean Functions", *Bell System Tech. Journal*, Vol. 35, No. 6, pp. 1417-1444, November 1956.

[7] C. Huitema, "Routing in the Internet", Prentice Hall, 1995.

[8] Steve E. Deering, "Host Extensions for IP Multicasting", RFC 1112, August 1989.

[9] Steve E. Deering, "Multicast Routing in Datagram Internetworks", Ph.D. Thesis, Stanford University, December 1991.

[10] H. Harney, C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, July 1997.

[11] H. Harney, C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, July 1997.

[12] A. Ballardie, P. Francis, J. Crowcroft, "Core Based Trees: An Architecture for Scalable Inter-Domain Multicast Routing", *Proceedings of the ACM SIGCOMM'93*, San Francisco, CA, September 1993.

[13] Ran Canetti, Juan Garay, Daniele Micciancio, Moni Naor, Benny Pinkas, "Issues in Multicast Security: A Taxonomy of Secure Multicast Protocols and Efficient Authentication Schemes", *draft manuscript*, 1998.

[14] Ran Canetti, Benny Pinkas, "A Taxonomy of Multicast Security Issues", *Internet Draft*, May 1998.

[15] Alan O. Freier, Philip Karlton, Paul C. Kocher, "The SSL Protocol Version 3.0", *Internet Draft*, *draft-freier-ssl-version3-02.txt*, November 1996.

[16] W. Yeong, T. Howes, S. Kille, "Lightweight Directory Access Protocol", RFC 1777, March 1995.

[17] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam, "Secure Group Communications Using Key Graphs", *Proceedings of ACM SIGCOMM*, Vancouver, British Columbia, September 1998.