A Lower Bound for Multicast Key Distribution

Jack Snoeyink, Subhash Suri, George Varghese

Abstract— **With the rapidly growing importance of multicast in the Internet there have been recent proposal, such as RFC 2627, for scalable key distribution such that when the nth user joins or leaves a group, broadcasting $\Theta(\log n)$ encrypted messages is sufficient to redistribute keys. In this paper, we show that this bound is also necessary for a general class of key distribution schemes and under different assumptions on user capabilities. While key distribution schemes can trade addition cost for deletion cost, for any scheme there is a sequence of $2n$ insertion and deletions whose total cost is $\Omega(n \log n)$. Thus, any key distribution scheme has a worst-case cost of $\Omega(\log n)$ either for adding or for deleting a user.**

Keywords—**Multicast, security, protocol analysis.**

I. Introduction

MANY distributed applications use a group paradigm, as people naturally work and interact in groups. Group types include one-to-many (e.g., pay-per-view or television broadcast) and many-to-many (e.g., distributed interactive games, teleconferencing, chat rooms). While applications using groups can be implemented over point-to-point communication links, (e.g., fiber links) there are advantages to using multicast or broadcast as the underlying communications primitive.

A broadcast channel allows a sender to communicate with every user that can listen to the channel using a single broadcast message. A satellite link, allowing a news source to broadcast to all users in the shadow of the satellite, is one example. Other examples include cable TV, microwave, and the Ethernet. With n users, broadcast can be n times cheaper than sending n separate unicast messages.

The notion of broadcasting extends to a network of point-to-point links, such as the Internet. In Fig. 1, for instance, source S can send a message to the group $N1 \ldots N4$, by sending a single copy of the message to the router, which can then make 3 copies of the message, one for the link to $N1$, one for the link to $N2$, and one for the broadcast link to which $N3$ and $N4$ are connected. Thus, messages are sent along a "Steiner tree," with routers making multiple copies of messages at branching points in the tree. This gives a bandwidth reduction from 4 to 1 on the common link from S to the router.

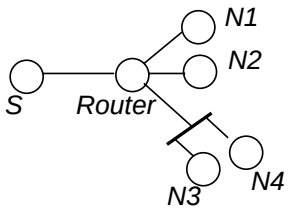

Fig. 1. Broadcast from S

An even more useful service is *multicast* where a message can be sent to a *subset* of all the Internet nodes. Thus in Fig. 1, S may wish to have a conference with $N1$ and $N2$ only. This is easily accomplished by assigning separate multicast addresses for each subset that wishes to communicate, and creating a separate Steiner tree for each such subset. One simple way to create a Steiner tree (used in the first Internet multicast experiments [6] on the MBONE[1]) is to first create a shortest path tree rooted at the source that spans all nodes, and then to prune away the links not required to reach the specified subset. For example, in Fig. 1, a tree connecting S to $N1$ and $N2$ can start with the broadcast tree and then prune away the link from the router to $N3$ and $N4$.

Although the MBONE is popular and can be used, for example, to hold conferences across the Internet, Internet Service Providers (ISPs) have historically resisted making their routers multicast-capable from fears of i) extra traffic due to multicast, ii) unstable multicast protocol implementations, and iii) the difficulty of implementing multicast charging models. Nevertheless, the remarkable effectiveness of multicast for distributing multimedia content (e.g., video) has led UUNET, a major ISP, to deploy multicast. Despite the slowness of initial deployment, multicast is likely to become an important and well-used Internet paradigm.

Multicast Key Distribution

The original Internet protocols paid little attention to secure communication, but commercial success has lead to many proposals for Internet security (e.g., IPsec [16]) that allow unicast messages to travel encrypted through the net-

The authors are at UNC Chapel Hill snoeyink@cs.unc.edu, UC Santa Barbara suri@cs.ucsb.edu, and UC San Diego vargese@cs.ucsd.edu. Work was done while the second and third authors were at Washington University in St Louis.

[1]The MBONE is a subset of the Internet consisting of multicast-capable routers and special logical "tunnel" links between them that are implemented using normal Internet routing

work. Security concerns for IP multicast are even greater, due to the nature and distribution of the traffic. When a multicast message is sent to one station on an Ethernet link, say $N4$ in Fig. 1, other stations, such as $N3$, can listen to the packets. If only $N4$ has paid for a service (e.g., video sent from S), then cryptographic techniques should prevent $N3$ from using the service.

This paper is about the problem of maintaining secrecy for multicast communication using any multicast or broadcast communication primitive, including the Internet multicast protocols as an important special case. Although there are proposals for group security that use sophisticated cryptographic techniques [14], we concentrate on secrecy by encrypted communication using simple and efficient private key techniques (e.g., DES) for group data encryption. Since secret key techniques are well studied and widely deployed, the main problem is key distribution: sending keys to all the group recipients in a *scalable* fashion.

The scalable key distribution problem is interesting because a number of applications can use *large* multicast groups that are also highly *dynamic* (i.e., users can be added or deleted frequently). RFC 2627 [17] describes two examples. The first is a distributed war gaming scenario that can have thousands of users participating at any time, with ten percent of the users changing over a period of one minute, and a constraint that users be added or dropped within a second. The second example, with similar parameters, is teleconferencing [17].

Simple extensions of unicast key distribution protocols (e.g., [9]) take linear time to add or remove a user, which would be problematic for the dynamic scenarios described above. Recent proposals [18], [5], [17] introduced a *Key Graph* scheme for scalable key distribution that takes $O(\log n)$ messages to add to or delete from a group of n users. We describe the Key Graph scheme in the next section. The main question that we investigate in this paper is whether the Key Graph scheme is optimal for scalable, multicast key distribution. For this, we must define the security requirements for key distribution.

Security requirements

Intuitively, the main requirement is *confidentiality*: only valid users should decrypt the multicast data even if the data is broadcast to the entire network. We assume in what follows that data is encrypted to ensure confidentiality using a symmetric cryptosystem such as DES. Thus, the confidentiality requirement can be translated into four requirements on key distribution: **Non-group Confidentiality:** First, users that were never part of the group should not have access to any key that can decrypt any multicast data sent to the group. **Future Confidentiality:** Second, we clearly require that users deleted from the group at some time t do not have access to any keys used to encrypt data after t unless they are added back to the group. **Collusion Freedom:** Third, no set of deleted users should be able to pool the keys they had before deletion to decrypt future communication. **Past Confidentiality:** Fourth, a user added at time t should not have access to any keys used to encrypt data before t while the user was not part of the group.

The last requirement is debatable. It protects against an attack in which an unsubscribed user could record encrypted broadcasts for a long period, then sign up for a short period to obtain decryption keys. Such an attack is unlikely for certain types of data and distribution channels: Stock quotes, for example, are unlikely to be recorded because they have value only when they are first broadcast. A cable television subscription is cheaper than the equipment to record encrypted broadcasts for a long period, especially since the shortest subscription period may be a month. (Perhaps one could record only the key exchange messages as they are broadcast, but encryption could make these indistinguishable from broadcast content.)

Therefore, in the hope of making key distribution easier, one could choose to drop the Past Confidentiality requirement or replace it by an assumption that users have no memory for past messages. Restricting the users' power allows the protocol more freedom. Somewhat contrary to the intuition, it makes it harder to establish a lower bound. The upper bounds in this paper show that if the first two requirement can be satisfied, then the third and fourth come easily. Models with and without Past Confidentiality have logarithmic lower bounds, but with different constant factors. Thus, dropping Past Confidentiality does not make it significantly easier to design secure key distribution schemes.

Paper Organization

The rest of this paper is organized as follows. In Section II, we review the relevant previous proposals for group security. In particular, we describe the Key Graph scheme in detail and briefly discuss a subsequent paper [4] that establishes a lower bound on the tradeoff between key storage and communication costs. In Section III, we describe graph models of a family of key distribution algorithms that distribute multiple keys per user but use a single key for data encryption. We move on in Section IV to discuss the difficulties of proving such a lower bound, and why it involves a tradeoff between add and delete costs. We prove our lower bounds in Section V; we conclude in Section VI by examining the significance of the results and

considering further extensions of our results.

II. Previous Work

We discuss previous work in the Internet milieu and in the theory community.

Internet Proposals

The simplest solution advocated in RFCs 2093 and 2094 for multicast key distribution is to assign a *user key* to each valid user, and one *group key* to the group [9], [8]. All keys are assumed to be keys from a symmetric cryptosystem such as DES in what follows unless other specified. Multicast data is encrypted by the group key. When a user u leaves, a new group key is chosen so that u cannot decipher future group communication. This is done by a group server who transmits the new group key to each remaining user u_i by encrypting with that user's key. Thus, deletion from a group of size n takes $(n-1)$ transmissions.

The Iolus system [11] attempts to improve the scalability of adds and deletes. The main idea is to establish a geographical hierarchy of keys using intermediaries called group security agents. When a user u_1 sends a message to the entire group, u_1 encrypts the message using its local key and sends the message to its local subgroup and to its group security agent. Each group security agent has access to the subgroup key of the subgroups it serves, and so can decrypt the packet and multicast the packet (re-encrypted using a server group key) on the network of group security agents. Each receiving server again decrypts the packet and re-broadcasts the packet to its local subgroup after re-encrypting using its local subgroup key. Thus, when a user u_i is deleted only the key of the subgroups that u_i belongs to need to be rekeyed.

The Iolus scheme has two problems. First, Iolus requires multiple encryptions during normal data operation. While the RFC 2093 and 2094 scheme requires only one encryption of a data packet, the Iolus scheme requires an encryption for each multicast server encountered in the multicast tree for the group. Since encryption is time consuming, it is preferable to make data operations (at gigabit rates) faster even at the cost of possibly slower add or delete operation. Second, Iolus requires multiple trusted entities (the group security agents), and each must work reliably (or be backed up) for multicast to work reliably.

In the last two years, several groups [3], [18], [5], [17], [13] have proposed variations on a technique to allow the use of a single group data key for data transmission (as in RFC 2093/2094) while having scalable add and delete operations (as in Iolus). Thus it is possible to have the best of both worlds. We use the term from Wong *et al.* [18], and refer to a *Key Graph* scheme in this paper.

The main idea is to have a single server as in RFC 2093/2094 but to have the server distribute subgroup keys in addition to the individual user keys and the group key. To balance the cost of additions and deletion, keys are arranged in a *logical* hierarchy (unlike the *physical hierarchy* of *Iolus*) with the root key being the root and the individual user keys being the leaves. The subgroup keys then correspond to the intermediate nodes of this conceptual tree. Intuitively, each subgroup key can be used to securely multicast to the users that are leaves of its corresponding subtree.

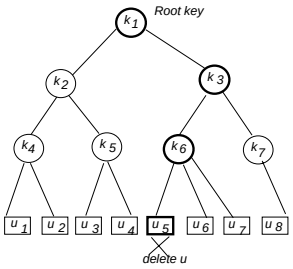

Fig. 2. An example of a logical tree underlying the Key Graph scheme. A group key server has distributed each node's key to every leaf in the subtree rooted at that node. In particular, the root key is known to all users and can be used to encrypt group data. The remaining keys support fast adds and deletes. The darker nodes need to be rekeyed when user u_5 is deleted.

Fig. 2 shows an example of a group of 8 users, u_1 to u_8, represented as leaves in what is almost a binary tree. While the tree can be arbitrary, we will see that balanced trees are required for fast adds and deletes. A group key server (not shown) has distributed each node's key to every leaf in the subtree rooted at that node. Thus, the root key k_1 is known to all the users, the subtree key k_3 to users $u_5 \ldots u_8$ etc. Each leaf also has the corresponding user key. Only the root key is used for data encryption, achieving similar performance for data encryption as in RFC 2093/2094. The subtree keys support fast rekeying.

Deletion and addition of users require maintenance operations on this logical tree. Deletion is accomplished by rekeying all the keys on the path from the deleted user to the root. (This is a good idea because, by definition, the deleted user has access to all these keys.) The only trick is to rekey from the bottom up, so that child subtrees have their keys for use when rekeying the parent. A balanced d-ary tree can be maintained at a cost of $\Theta(d \log_d n)$ encrypted messages per operation. It is easy to show that $d \log_d n$ is minimized when d is 3.

Previous Work in the Theory Community

For several years, the theory community has studied the problem of allowing only legitimate users to access multicast data. Starting with Fiat and Naor [7], and continuing with [10], [2], [15], the theory community has studied this problem using different models and different assumptions from those used in the Internet community. We refer the reader to [4] where these differences are well treated. For example, the Fiat-Naor [7] paper does not require a single key for data sent to the group, and does not require any action after adds and deletes; however, it makes assumptions about the possible coalitions formed by users outside the multicast group. Luby and Stinson [10] allow arbitrary coalitions but restrict the possible types of multicast groups.

The results most closely related to ours is work by Canetti, Malkin and Nissim [4]. They describe a modification of the RFC 2627 protocol using pseudo-random number generators and other devices to trade between storage and communication costs. They establish a lower bounds on this tradeoff, based on a limit to the number of keys held by a users. If each user holds at most b keys, then the communication costs are at least $n^{1/b} - 1$, and if the protocol is from for a special class of *structure preserving protocols*, which includes their protocol, the bound can be strengthened to $bn^{1/b} - 1$. Thus, if a user is limited to $\log n$ keys and a structure-preserving protocol, the communication costs will be $\Theta(\log n)$.

We conclude that the Key Graph scheme is of practical interest for the Internet because of its simplicity and scalability (except for server storage which may not be currently a practical issue). When evaluating broadcast key distribution schemes, we agree with Canetti *et al.* [4] who state that "communication complexity is probably the biggest bottleneck in current applications. (Indeed, removing communication is the main motivation for using multicast technology.)" Thus, we believe an important question is whether the Key Graph scheme is communication optimal. The lower bound in [4] treats a different problem. To answer our question, we must have a general model of single data key encryption schemes and their accompanying key distribution mechanisms. We now turn to proposing such a model, which we then use to prove our lower bounds.

III. Base Model

We start with essentially the model of Canetti *et al.* [4], which we then translate into a graph model that is more convenient for the lower bound proof. Define the *multicast group* $M = \{u_1, u_2, \ldots u_n\}$, which is a (dynamically changing) subset from a universe of all possible users, and a *central server* $s \notin M$, called the center in [4]. We assume that any message sent to the multicast group can be received by all current members of M and possibly by other users not in M. Thus all data sent to the multicast group is encrypted using a group key k_M that is shared by all current members of M.

To abstract away the secret key cryptographic details, following Canetti *et al.* [4], we assume there is publically available black-box pair E, D that takes a message m and key k as input and outputs a random ciphertext $c = E(k, m)$; given ciphertext c and key k, $D(k, c) = m$. Thus any user holding k will be able to decrypt but no coalition of users not holding k will be able to decrypt or gain any information, even assuming a computationally unbounded adversary. We must, of course, assume that users cannot distribute plaintext keys, or that such distribution can be detected by infrequent polling.

A *multicast protocol* specifies an algorithm by which the server s can update the group key (and possibly other keys) for two operations of $\textsc{Add}(u)$ for $u \notin M$, which results in the multicast group changing from M to $M \cup \{u\}$ and $\textsc{Delete}(u)$ for $u \in M$, which results in the multicast group changing from M to $M \setminus \{u\}$.

We will study a more specific model called *key-based multicast protocols* [4]. Let l be a security parameter that is polynomial in the number of users n. Let $K \subset \{0,1\}^l$ be a set of keys. Each user $u_i \in M$ holds a subset $K(u_i) \subseteq K$ of keys; $|K(u_i)| \geq 2$ for all i, as every user holds the group key k_M and an individual key k_i.

For security, we consider a computationally unbounded adversary who can repeatedly submit update operations in any order and have access to keys belonging to users that are not currently part of M (but not to users currently in M). A key-based multicast protocol is *secure* if for any adversary, after any sequence of operations, the adversary has no way to distinguish k_M from a random key. This definition provides past, future, and non-group confidentiality, and assures collusion freedom.

The model must handle two updates operations. To handle an $\textsc{Add}(u)$ operation, the central server can broadcast the group key encoded with the individual key for u. To handle a $\textsc{Delete}(u)$ operation, the server simply ceases to use the keys held by u. In both cases, the server may also broadcast additional messages that result in one or more users gaining new keys. Thus, it will be important that our model allow us to determine which keys a user holds.

We measure the communication cost of an update in terms of the number of encrypted messages broadcast by a server. We assume, for any two keys k_1 and k_2, that $E(k_1, k_2)$ is a message of c bits. Notice that the single

broadcast $E(k_1, k_2)$ will send the key k_2 to exactly the set U of users that hold the key k_1. We focus on the worst case update complexity for rekeying when adding or deleting users.

Since unsubscribed users may still receive multicast messages, we can consider whether they can have *memory*; whether or not they can save key distribution messages for later decoding, should they come into possession of the appropriate keys. This makes a difference in our models and bounds.

IV. Lower Bound Approaches

We briefly discuss why there must be a tradeoff between delete and add costs by giving three examples. Then we outline the difficulty of extending the lower bound approach of Canetti *et al.* [4] to find an absolute lower bound on communication costs.

First, suppose that when a new user u_i joins a multicast group M', we distribute keys for all possible subsets of users in the set $M = M' + u_i$. Clearly, this addition scheme would broadcast exponentially-many keys. Deletion of user u_j, however, can now be done by simply choosing a new group key and sending it to $M - u_j$ using a single encryption/multicast. Each node can discard all keys held by subsets containing u_j, so that they will never again be used. Thus the cost of adding a user is exponential, but deletion is $O(1)$.

Asking n users to each store 2^n keys is clearly too much. For a second example at the other extreme, suppose that each user u_i can hold only two keys, the group key and the individual user key. If some user u_i leaves M, then the server can only use the individual user keys to send the new group key to $M - u_i$, and thus the delete cost is $O(n)$.

Third, the Key Graph scheme [18], [17] takes $\Theta(\log n)$ for both add and delete costs. These examples suggest that algorithms can trade add complexity for delete complexity or vice versa. In particular, we aim to show that if the worst-case delete (add) complexity is less than logarithmic cost, then the corresponding add (delete) complexity is at least logarithmic cost. This suggests we need to argue about sequences and not just about isolated updates.

Canetti *et al.* [4] use the maximum number of keys stored by a user to investigate delete costs. In one example they suppose each user has at most 3 keys: $|K(u_i)| \leq 3$ for all i. Let X be the largest subgroup other than M and let $x = |X|$. Suppose a user u_i is deleted from X. Then to send the new global key to all users outside of X, we must pay at least $(n - x)/x$ encryptions (since each user shares a key with at most x other users by assumption). To send the new global key to all users remaining in X, we must pay $x - 1$ encryptions because by assumption, each user in X is a member of at most 2 groups other than than the multicast group; thus we must use individual encryptions to reach the members of X. The total cost is $x - 1 + (n - x)/x$ which is minimized when $x = \sqrt{n}$ for an overall minimum cost of $2\sqrt{n} - 2$.

One might hope to extend this argument to the case when every user has at most four keys as follows. Again let X be the largest set other than M. Once again, if someone leaves within X we must pay $n - x/x$ to reach users not in X. However, within X is now an instance of the problem where each user has at most 3 keys. From the above result, we have an overall cost of $2\sqrt{x} - 2 + n - x$. After minimizing, the result is a minimum cost of $3n^{1/3} - 3$. We might hope that the argument would generalize to show that if each user has at most j keys, the delete cost is at least $jn^{1/j} - j$. This in turn would imply that if $j = \log n$, then the delete cost is $\Theta(\log n)$.

The flaw in the above argument for $j \geq 3$ is that it assumes that subsets do not overlap (as in the RFC 2627 algorithm). But they may overlap in general, so that sending the new global key to all users not in X could also be sending the key to some c users in X. The simple argument now breaks down. This is why [4] introduces the extra assumption of structure preserving protocols to prove such a result. We would like a lower bound without the assumption that the protocol is "structure preserving" and without a bound on the number of keys held by any user.

V. Proof of Lower Bound

We prove a logarithmic lower bound on key distribution. In Section V-A we translate the problem for users with memory into a graph model; this reduces the lower bound to an argument about graph properties. In Section V-B we provide a lower bound on the cost of communications, based on the cost of deleting edges from the graph model. In Section V-C, we extend the graph to model users without memory, and in Section V-D we extend the lower bound.

A. Representing Protocol State by a Broadcast History Graph

To reduce the lower bound to an argument about graph properties, we translate the base model into one using graphs. A natural model used by existing protocols is a directed graph whose nodes are in one-to-one correspondence with keys. In fact, we use *node k* to mean the node with key k. Each user is represented by the node containing his or her individual key. Since distribution of individual keys is not permitted, user nodes have outdegree zero and will occasionally be called the *leaves* in this directed graph.

Let *users*(k) denote the set of user nodes for users who have a copy of k. Wong, Gouda, and Lam [18] defined a *key graph* as a bipartite graph with an edge from a key k to every user node in *users*(k). A more hierarchically structured representation can be defined as follows: the children of node k are all nodes for keys l such that *users*$(l) \subset$ *users*(k) and there is no other key m such that *users*$(l) \subset$ *users*$(m) \subset$ *users*(k). In other words, k's children are the keys whose user sets are the maximal subsets of k's user set. This structured representation is more compact, but still has the property that, from every key k, there is a path in the directed graph to each node in *users*(k). There is always a root node that represents the group key, and has paths to all user nodes.

The difficulty with these state representations is that they keep no trace of past communications used to distribute the keys. Thus it is harder to infer past communication costs from the structure of the graph.

Example 1: Suppose that the state graph is as shown on the left in Fig. 3. Users u_1 and u_2 share a group, created by say sending k to both u_1 and u_2 using their individual keys. Suppose now we add a new user u_3 and update the group state by first creating a new individual key for u_3, and then creating a new group key l and sending it individually to u_1, u_2 and u_3. The state graph changes to the one shown on the right in Fig. 3. This is because l is now shared by all 3 users, but *users*$(k) \subset$ *users*(l). Although the algorithm sent three messages to distribute l, the simple state graph gains just two extra edges, one between l and k and one between l and u_3.

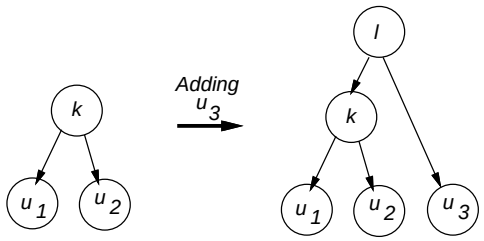

Fig. 3. How the state of a protocol shown on left changes after adding user u_3 in the simple state graph (new state on right). Notice that the actual communication is hidden because edges indicate subset relationships.

We prefer a *broadcast-based history graph* (see Fig. 4) in which nodes corresponds to currently valid keys, and in which there is an edge from node l to node k if and only if $E(k, l)$ is broadcast, sending key l encrypted by key k. Using this basic model, the state of the same protocol before adding u_3 as described is shown at the left in Fig. 4 and the state after adding u_3 is shown at right. Notice that 3 edges have been created, recording the actual communication costs, unlike the simple state model in Fig. 3. As before, the user keys are *leaves*, and there is always a distinguished root that represents the group key (e.g., for the two examples of Fig. 4, the root is k at left and l at right.

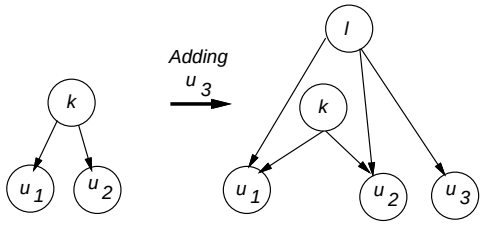

Fig. 4. How the state of a protocol shown on left changes after adding user u_3. Notice that the actual communication costs incurred so far is recorded in the number of edges in the current graph.

We would like to claim that the set of users who could possibly decrypt a message sent encrypted by key k are exactly those users with a directed path from node k. As we will see in the next example, the truth of this claim depends on whether or not the users have memory for past messages.

Example 2: Suppose that a key k is created and sent to two users u_1 and u_2 using their individual keys. Next a new key l is broadcast encrypted by k (and thus is read by u_1 and u_2). The state of the broadcast history graph is shown on the left of Fig. 5. So far so good. Suppose now u_3 is added and assigned an individual key. Then the update protocol has the server send the old key k encrypted by u_3's individual key. Thus our broadcast history graph becomes as shown on the right of Fig. 5. Notice that there is a path between l and u_3 although u_3 does not directly receive a copy of l.

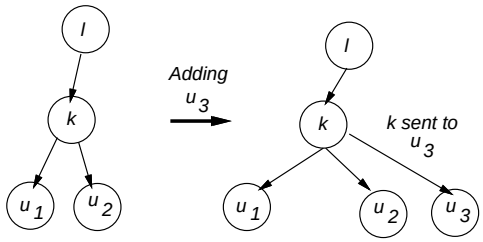

Fig. 5. Sending key k to u_3 creates the graph on the right where l has a path to u_3 but u_3 has no copy of key l.

Suppose that user u_3 may have sufficient memory to have recorded broadcast messages received before u_3 joined the group. To be safe, it must be assumed that u_3 can recover key l: The arrow from l to k indicates that sometime in the past $E(k, l)$ was broadcast, and u_3 might have recorded that broadcast for later playback when he or she received key k. Thus, for users with memory, the basic

broadcast history graph accurately models the set of users that could hold key l as those with a directed path from node l.

On the other hand, for users without memory, a path between a key node k and a user u_i does not imply that u_i has a copy of k. This is problematic because our lower bound proof will rely on the fact that the root node (group key) is always connected to all users. By looking only at the graph, one cannot determine what users hold what keys. We will modify the model to reflect users with no memory in section V-C.

B. A lower bound for users with memory

To establish our lower bound, we first need an intermediate result on the number of edges that must be deleted in the broadcast history graph when a user departs from the multicast group. This does not directly imply a lower bound on actual communication costs because these edges could have been created by *any* past update operation. However, we will exploit this result later in the section by arguing about sequences.

For a user node u_i in the history graph G, define the *ancestor weight* w_i as the sum of the outdegrees of all nodes on paths from the root to u_i. (By the problem definition, we have at least one path because u_i must have received the group key.) For example, if the history graph is a binary tree, the ancestor weight of any leaf is twice its depth. Let w_G denote the maximum ancestor weight over all user nodes u_i of of G, and let $w(n)$ denote the minimum value of $w(G)$ over all history graphs G with n leaves.

Lemma 1: The ancestor weight of a leaf u_i is the number of edges that will disappear from the history graph when the user corresponding to this leaf is deleted.

Proof: Each node on a valid path to leaf u_i corresponds to a key k that u_i holds. To preserve future confidentiality, k cannot be used after u_i is deleted. ■

Minimum ancestor weight is achieved by certain trees.

Lemma 2: The minimum ancestor weight for a broadcast history graph of n leaves satisfies $w(n) \geq \lceil 3 \log_3 n \rceil$. This is attained by a 2–3 tree.

Proof: Note that the weight $w(n)$ is an nondecreasing function of n: for any broadcast history graph with $n > 1$ leaves, we can merge two leaves without increasing ancestor weight, so $w(n-1) \leq w(n)$.

We can transform any history graph G, without increasing ancestor weight, into a *2–3 tree* in which every node has outdegree two or three. First, choose a directed tree in G with paths to all leaves. A simple graph traversal shows that this is always possible, since there are paths from the root to every user node. We can omit all edges not in the spanning tree, since this can only decrease the number and degrees of ancestor nodes of any leaf.

Second, replace any node of outdegree greater than three with a binary tree. This replaces a degree-k node by a path of at most $\lceil \log k \rceil$ degree-two nodes, and $k \geq 2\lceil \log k \rceil$ for all $k > 3$. Third, merge each node of outdegree one with its parent.

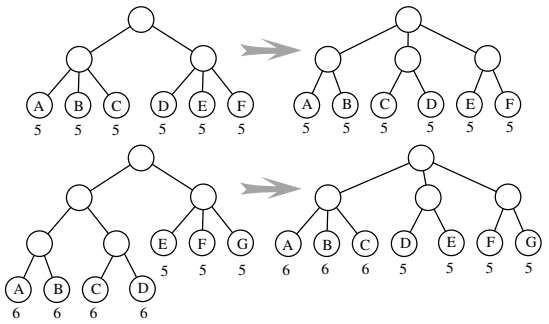

Fig. 6. Example transformations eliminating the lowest degree-2 parent of a degree-3 node without increasing weight. Numbers are ancestor weights of the labeled leaves.

A final transformation moves all the degree-two nodes to the lowest levels of the tree. Repeatedly replace the lowest degree-two node ν that is a parent of a degree-three node as follows: if ν has two children of degree three, replace them with a degree three parent of degree two children, as in the top half of Fig. 6. Otherwise ν has one binary subtree, a portion of which can be replaced by a shorter 3-ary tree as in the lower half of Fig. 6, without increasing the maximum ancestor weight.

Now, consider the maximum number n of leaves in a tree of a given weight $w(n) = k$; since $w(n)$ is an nondecreasing function on integers, this information determines the function. We establish by induction that the number of leaves for trees of weight $3i$, $3i+1$, and $3i+2$ are 3^i, $4 \cdot 3^{i-1}$, and $2 \cdot 3^i$, respectively.

In the base cases, the maximum number of leaves for trees of weight 3, 4, and 5 are, respectively, 3, 4, and 6. (This last tree is in the top of Fig. 6.) In the induction step, we form a tree by adding a root to the best subtrees. We can observe that the root must be degree three unless the entire subtree below is binary. But this happens only for the weight 4 tree on 4 leaves; at weight 6, the binary tree with 8 leaves is dominated by the ternary tree with 9 leaves. All roots after the base case must have degree 3, therefore, and the induction holds. We can summarize by saying that $w(n) \geq \lceil 3 \log_3 n \rceil$. ■

By Lemma 1, the ancestor weight of a leaf is the number of edges that will disappear from the broadcast history graph when the user corresponding to this leaf is deleted. Thus, if an adversary deletes the node with largest weight,

we know it must have deleted at least a logarithmic number of edges.

Note that bounding the edges deleted after any DELETE(u) operation does not imply anything about the communication costs incurred for this particular operation. In particular, it does not follow that every DELETE(u) operation must cost $\Theta(\log n)$ in communication complexity. Of course, we do know that each edge did cost c bits of communication to create in the past.

It is tempting to conjecture that the cost of the deleted edges when u_i is deleted is sum of the real communication costs to add user u_i and to delete user u_i. In that case, at least one such cost will be logarithmic (by our last lemma) and we would be done. This, however, is not true. When u_i is deleted it may have paths from several keys, say k_1 through k_m, and not all these nodes may have been created when u_i was added. For example, when u_i was created it may have only had a valid path to say k_1. The other nodes and valid paths may have been added as subsequent users were added. (This is true for even the RFC 2627 algorithm.)

Thus, all we know is that the logarithmic number of edges deleted after every delete must have been created *in some past update operation*. This is too weak to prove a lower bound on the particular delete operation. However it is sufficient to prove a lower bound on the costs of a specific sequence of updates, as shown in the following theorem.

Theorem 3: For any secure multicast protocol, there is a sequence of $2n$ ADD and DELETE operations such that the total communication costs is $\Theta(n \log n)$ encrypted messages.

Proof: Consider a sequence of n insertions of users u_i through u_n and then a sequence of deletions of all the users, where at each stage the user with maximum ancestor weight is deleted. By Lemma 2, the total number of edges deleted must be $\geq 3(\log_3 n + \log_3(n-1) + \log_3(n-2) + \ldots \log_3 1)$. Thus the number of edges is $\geq 3\log_3(n!) = \Theta(n \log n)$, Since each of these edges must have been created in the past at a cost of a constant c bits of encrypted communication, the total communication cost over this sequence of n operations is $\Theta(n \log n)$. ■

C. Annotating the broadcast history graph for users without memory

For users without memory of previous key distribution messages, this lower bound proof does not apply! Recall that the model of section V-A assumes that a user u holds each key k for which there is a directed path in the broadcast history graph from k to u. But if u has no memory, then the path may have been created after k was broadcast, as it was in Fig. 5. Restricting the users' capabilities gives more information to the protocol and makes lower bounds harder to establish. In the rest of the paper, we re-establish the lower bound with slightly smaller constants.

First, we change the model by adding a little bit of information to every edge. We give every broadcast message event a sequentially increasing time stamp in the order they were sent by the server. We then label any edge with the time stamp of the broadcast event that caused this edge to be formed. This permits cycles and multiple edges between nodes, and supports the modeling of quite unstructured protocols.

In this graph, a *valid path* is a directed path such that the time stamps on all consecutive edges are non-increasing. This graph will always have the property that the user nodes that receive a copy of key k are those reachable by a valid path from node k. It is not hard to show inductively that this is true when an edge from k to l is created in the graph by a broadcast of k encrypted by l. The protocol must maintain that there is some valid path from a distinguished root, holding the group key, to every user node.

Thus if we revisit Example 2 with an annotated history graph we get the two snapshots shown in Fig. 7 for the state before (left) and after (right) adding u_3. We can easily tell that u_3 does not have a copy of l from the extra information because there is no valid path from l to u_3. While there is a directed path from l to u_3, since the first edge has time stamp 3 and the last edge has time stamp 4, the time stamps increase and this path is not a valid path.

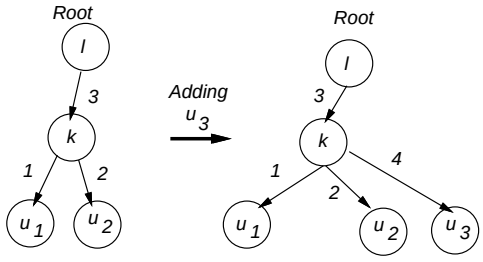

Fig. 7. Adding time stamp information allows us to deduce that u_3 does not have a copy of l because there is no path with non-increasing time stamps from l to u_3.

To update the annotated history graph when a user u_i is deleted, we first delete the leaf user node and delete all key nodes (and their incident and outgoing edges) that have valid paths to u_i. These keys cannot be used for future communication without compromising future confidentiality, because u_i has those keys. For example, if we delete u_1 from the right of Fig. 7, the update must first start by deleting nodes k, l, and u_3 and their edges; only then will new nodes be added. We note that the update algorithm may not actually pay for any broadcasts to delete

these keys at that stage (or to replace them); however, they cannot be used for future broadcasts. Similarly, note that after u_i is deleted and we remove the key nodes that u_i possesses, an algorithm can create an arbitrary number of key nodes and broadcast them using other existing key nodes, thus creating edges. Notice that it would be hard to define which edges should be deleted in the basic broadcast history model as opposed to the annotated model. A formal model of how the history model is updated is in an appendix.

D. A lower bound for users without memory

With some complications due to the timestamps, we can use the annotated history graph to prove a lower bound as in Section V-B.

When a user corresponding to a leaf u_i departs from the multicast group, it holds each key k on valid paths from the root to u_i. Key k cannot be directly used again, nor can any key that was broadcast encoded by k while u_i held k; these edges must disappear from the annotated history graph. Distributions that were encrypted using k before u_i held k remain safe, however, if u_i is assumed to have no memory.

We therefor define *valid ancestor weight* as follows: for each node, determine the minimum timestamp (which may be $+\infty$) that can be extended to a valid path to u_i. Charge that node for all edges with greater or equal timestamps. The *valid ancestor weight for a graph* is again defined as the maximum leaf weight. We briefly sketch a bound on the minimum valid ancestor weight for a graph of n leaves.

Lemma 4: The minimum valid ancestor weight for an annotated history graph of n leaves satisfies $w(n) \geq \lceil \log_2 n \rceil / 2$.

Proof: Valid ancestor weight is again minimized by certain trees: since there are valid paths to all leaves, we can form a spanning tree of valid paths without increasing the weight of any graph. We can assume that at each node the timestamps on edges to children increase from right to left. For any node with a single child, we contract the earlier of the two incident edges to merge that node into its parent or child.

We transform an annotated history tree into a binary tree by taking each node of degree $k > 2$ and replacing it with a left-slanting tree of $k - 1$ nodes; all children after the first become right children of one of these nodes. The valid ancestor weight that the original node contributed to its children is the depth of each child, except for the first child, whose valid ancestor weight was one more than its height. The maximum in the resulting binary tree of n leaves is simply the tree height, which is at least $\lceil \log_2 n \rceil$. ■

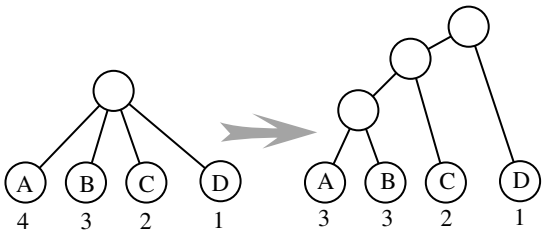

Fig. 8. Transforming a spanning tree with timestamps into a binary tree so that the original valid ancestor weight is bounded from below by node depth.

As before, we cannot bound the cost of an individual addition or deletion, but we can bound the cost of a sequence.

Theorem 5: For any secure multicast protocol, there is a sequence of $2n$ ADD and DELETE operations such that the total communication costs is $\Theta(n \log n)$ encrypted messages.

Proof: Consider a sequence of n insertions of users u_i through u_n and then a sequence of deletions of all the users, where at each stage the user with maximum valid ancestor weight is deleted. By Lemma 4, the total number of edges deleted must be $\geq (\log_2 n + \log_2(n-1) + \log_2(n-2) + \ldots \log_2 1)$. Thus the number of edges is at least $\log_2(n!) = \Theta(n \log n)$, Since each of these edges must have been created in the past at a cost of a constant c bits of encrypted communication, the total communication cost over this sequence of n operations is $\Theta(n \log n)$. ■

VI. Conclusion and Open Questions

We have shown that the logarithmic factor that appears in secure group key maintenance schemes such as RFC 2627 [17] is necessary under the assumption that a single message contains a single key and assuming that we restrict ourselves to key-based multicast protocols. In order to obtain a protocol with sub-logarithmic update costs, therefore, one would need to use a different model. One approach is to relax the security constraints, allowing delays for users joining or leaving the group as in the Kronos system proposed by Setia et al. [12], which allows users who leave the group to receive content until a rekeying period. Another approach may be to "look under the cryptographic hood" and use other assumptions besides secret key cryptography. It would be nice to generalize our lower bound to fit a more general cryptographic model. Finally, it would be nice to explore how limits on the number of key broadcasts on deletion of a user translate to more key broadcasts; for example, if deletion from a group of size n must be accomplished in one message, then keys must be maintained for all subsets of size $n - 1$.

We wish to thank a reviewer for comments that lead to clarifying the distinction between users with and without

memory.

REFERENCES

[1] A. Ballardie. Scalable Multicast Key Distribution. *RFC 1949*, May 1996.

[2] C. Blundo, L. Mattos, and D. Stinson. Tradeoffs between communication and storage in unconditionally secure schemes for broadcast encryption and key distribution. *Advances in Cryptology — CRYPTO 96.*, 1996.

[3] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas. Multicast Security: A Taxonomy and Efficient Authentication. *Proc. of INFOCOM'99*, March 1999.

[4] R. Canetti, T. Malkin, K. Nissim. Efficient Communication-Storage Tradeoffs for Multicast Encryption. *Advances in Cryptology, EUROCRYPT 1999*, May 1999.

[5] Germano Caronni, Marcel Waldvogel, Dan Sun, Bernhard Plattner Efficient Security for Large and Dynamic Multicast Groups. *Proc. Seventh Workshop on Enabling Technologies, (WETICE '98)*, IEEE Computer Society Press, 1998.

[6] S. Casner and S. Deering. First IETF Internet Audiocast. *SIGCOMM Computer Communication Review*, July 1992.

[7] A. Fiat and M. Naor. Broadcast Encryption. *Advances in Cryptology — CRYPTO 93.*, 1993.

[8] H. Harney, C. Muckenhirn, and T. Rivers. Group Key Management Protocol Architecture, *RFC 2094*, September 1994.

[9] H. Harney, C. Muckenhirn, and T. Rivers. Group Key Management Protocol Specification, *RFC 2093*, September 1994.

[10] M. Luby and J. Staddon. Combinatorial Bounds for Broadcast Encryption. *Advances in Cryptology — EUROCRYPT 98.*, 1998.

[11] S. Mittra. Iolus: A framework for scalable secure multicasting. *Proc. of ACM SIGCOMM '97*, pages 277-288, September 1997. STW97

[12] S. Setia, S. Koussih, S. Jajodia and E. Harder. Kronos: A Scalable Group ReKeying Approach for Secure Multicast. *IEEE Symposium on Security and Privacy*, pp. 215-228, 2000.

[13] C. Shields and J. Garcia-Luna-Aceves. KHIP - a scalable protocol for secure multicast routing. In *Proceedings of ACM SIGCOMM'99*, 1999.

[14] M. Steiner, G. Tsudik, and M. Waidner. Cliques: A protocol suite for key agreement in dynamic groups. *Proc. of ICDCS'98*, Amsterdam, May 1998. Also *Research Report RZ 2984 (93030)*, IBM Zurich Research Lab, December 1997.

[15] D. Stinson and T. van Trung Some new results on key distribution patterns and broadcast encryption *Designs, Codes and Cryptography*, to appear.

[16] R. Thayer, N. Doraswamy, and R. Glenn. IP Security Document Road Map. *RFC 2411*, Nov 1998.

[17] D. Wallner, E. Harder, and R. Agee Key Management for Multicast: Issues and Architectures *RFC 2627*, June 1999.

[18] C. Wong, M. Gouda and S. Lam. Secure Group Communications Using Key Graphs. *Proceedings SIGCOMM 98*, Sept 1998.

APPENDIX

I. FORMAL MODEL FOR UPDATING BROADCAST HISTORY GRAPH

Thus for any key-based multicast protocol, we will change the state of the annotated history graph as shown in Fig. 9 in the Appendix. Note that we assume there is always a root node that corresponds to the group key and that there is always a valid path between the root node and the set of current users. (Thus for example the right half of Fig. 7 is an inavlid state.) We use CREATEEDGES to represent the creation and broadcasting of new keys by the algorithm. We only require that after it is finished there is a valid path between root and all user nodes. This cannot be trivially satisfied after a DELETE because a DELETE will begin by deleting the existing root node.

INITIALIZATION:
  oldKeySet = *nil*; (* keeps track of old keys to avoid repeats*)
  timeStamp = 1;
  root = *nil*; (* no group key initially *)

ADD(u_i) (* user i is added to multicast group *)
  Create a node u_i using a key not in *oldKeySet*
  CREATEEDGES(* create new nodes and edges *)

DELETE(u_i) (* user i is deleted from multicast group *)
  Delete all nodes including u_i that have a valid path to i
  Add all deleted node keys to *oldKeySet*
  Delete all edges outgoing or incident on such deleted nodes.
  CREATEEDGES (* create new nodes and edges *)

CREATEEDGES (* subroutine to change graph *)
  Create a possibly empty set of new nodes $k_1 \ldots k_m$
  Label each node k_i with a key not in *oldKeySet*
  Add new edges non-deterministically between nodes so that
    Each created edge is labelled with *timeStamp*
      and *timeStamp* increments
    No edge created from a user node (user key is private)
    During process, *root* can change to any graph node.
  After all edges added, *root* has a valid path to all user nodes

Fig. 9. Code to update annotated broadcast history graph