Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Networks 47 (2005) 429–441
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

A lower bound for multicast key distribution

Jack Snoeyink [a], Subhash Suri [b], George Varghese [c,*]

[a] Department of Computer Science, UNC-Chapel Hill, Chapel Hill, NC 27599-3175, USA
[b] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
[c] Computer Science and Engineering Department, University of California, 9500 Gilman Drive, La Jolla, CA 92093-0114, USA

Received 2 September 2002; received in revised form 3 March 2004; accepted 7 September 2004
Available online 19 October 2004

Reponsible Editor: S. Lam

Abstract

With the rapidly growing importance of multicast in the Internet, several schemes for scalable key distribution have been proposed. These schemes require the broadcast of $\Theta(\log n)$ encrypted messages to update the group key when the nth user joins or leaves the group. In this paper, we establish a matching lower bound (Independently, and concurrently, Richard Yang and Simon Lam discovered a similar bound with slightly different properties and proofs. An earlier version of our paper appeared in Infocom 2001 while their result appears in [R. Yang, S. Lam, A secure group key management communication lower bound, Technical Report TR-00-24, Department of Computer Sciences, UT Austin, July 2000, revised September 2000].), thus showing that $\Theta(\log n)$ encrypted messages are necessary for a general class of key distribution schemes and under different assumptions on user capabilities. While key distribution schemes can exercise some tradeoff between the costs of adding or deleting a user, our main result shows that 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

1. Introduction

Many distributed applications—such as interactive games, teleconferencing, and chat rooms—use a *group* paradigm. While such applications using groups can be implemented over point-to-point communication links, there are advantages to using multicast or broadcast as the underlying communications primitive.

A broadcast channel such as a satellite allows a sender to communicate with every user that can listen to the channel using a *single* broadcast

* Corresponding author. Tel.: +1 858 822 0424.
E-mail addresses: snoeyink@cs.unc.edu (J. Snoeyink), suri@cs.ucsb.edu (S. Suri), varghese@cs.ucsd.edu (G. Varghese).

1389-1286/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2004.09.001

message. 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 where the routers can make extra copies of a message for all downstream links to which the message is intended. Further, broadcast generalizes to *multicast* where a message can be sent to a *subset* of all the Internet nodes.

Multicast is easily accomplished by assigning separate multicast addresses for each subset that wishes to communicate, and creating a separate Steiner tree [7] for each such subset. Despite the slowness of initial deployment, Internet multicast [7] is likely to become an important and well-used Internet paradigm.

The original Internet protocols paid little attention to secure communication, but commercial success has lead to many proposals for Internet security (e.g., IPsec [18]) that allow unicast messages to travel encrypted. 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 say a satellite link, other stations in the range of the satellite can listen to the packets. If the listening stations have not paid for the service, then cryptographic techniques must be used to prevent unauthorized listeners 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 [16], 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). For example, distributed war gaming and teleconferencing [19], applications can have thousands of users 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.

Simple extensions of unicast key distribution protocols (e.g., [10]) take linear time to add or remove a user, which would be problematic for the dynamic scenarios described above. Recent proposals [20,6,19] 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.

1.1. Security requirements

Intuitively, the main requirement is *confidentiality*: only valid users should be able to 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 the following four requirements on key distribution:

Non-group confidentiality: 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: Users deleted from the group at 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: No set of deleted users should be able to pool the keys they had before deletion to decrypt future communication.
Past Confidentiality: 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 per-

iod, 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, have value only when they are first broadcast. Fortunately, the bounds in this paper show that if the first two requirements can be satisfied, then the third and fourth come easily. Specifically, we show that the models with and without Past Confidentiality have the same logarithmic lower bounds, but with different constant factors. Thus dropping Past Confidentiality as a requirement does not affect our results significantly.

Paper organization: The rest of this paper is organized as follows. In Section 2, 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 [5] that establishes a lower bound on the tradeoff between key storage and communication costs. In Section 3, 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. In Section 4, we 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 5; we conclude in Section 6 by examining the significance of the results and considering further extensions of our results.

2. Previous work

We discuss previous work in the Internet and the theory communities.

Internet proposals: The simplest solution advocated in RFCs 2093 and 2094 for multicast key distribution is to assign a *user key* for each valid user, and one *group key* for the group [10,9]. When a user u leaves, the new group key is sent to each remaining user u_i by encrypting with that user's key using $(n-1)$ transmissions. The Iolus system [13] improves scalability of adds and deletes using a geographical hierarchy of keys. Unfortunately, Iolus requires time-consuming encryption for several multicast server encountered in the path, and requires multiple trusted entities (the so-called group security agents).

In the past few years, several groups [1,4,20,6,19,15] 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. [20], and refer to a *Key Graph* scheme in this paper.

The main idea is to have a single server but to have the server distribute subgroup keys in addition to the individual user keys and the group key. Keys are arranged in a *logical* hierarchy 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. Each subgroup key can be used to securely multicast to the users that are leaves of its corresponding subtree.

Fig. 1 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, balanced trees are required for fast adds and deletes. The root key k_1 is known to all the users, the subtree key k_3 to users $u_5, \ldots, u_8$, etc. 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 is accomplished by rekeying all the keys on the path from the deleted user to the root. The main idea 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

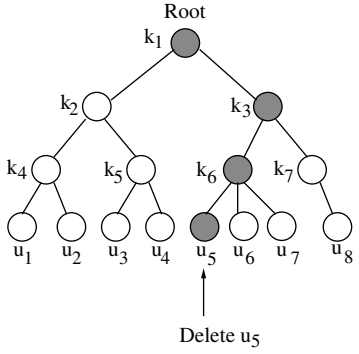


Fig. 1. An example of a logical tree underlying the Key Graph scheme.

messages per operation. It is easy to show that $d\log_d n$ is minimized when d is 3.

2.1. Previous work in the theory community

Starting with Fiat and Naor [8], and continuing with [11,3,17], the theory community has studied the multicast security problem using different models and different assumptions from those used in the Internet community. We refer the reader to [5] where these differences are well treated.

The results most closely related to ours is work by Canetti et al. [5]. They describe a modification of the RFC 2627 protocol using pseudo-random number generators and other devices to achieve a tradeoff between storage and communication costs. They establish 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 $\Theta(\log n)$.

When evaluating broadcast key distribution schemes, we agree with Canetti et al. [5] who state that "communication complexity" is probably the biggest bottleneck in current applications. Thus, we believe an important question is whether the Key Graph scheme is communication optimal. The lower bound in [5] treats a different problem. By contrast, we do not restrict ourselves to structure-preserving protocols, and show that irrespective of the number of keys held by users (as long as it is not exponential), the communication cost is at least $\log N$.

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.

3. Base model

We start with essentially the model of Canetti et al. [5], 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 [5]. 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. [5], we assume there is publically available black-box pair E, D that takes as input a message m and key k and outputs a random ciphertext $c = E(k, m)$; given ciphertext c and key k, the box produces the original message $D(k, c) = m$. 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* [5]. 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; $|K(u_i)| \geqslant 2$ 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 ADD(u) operation, the central server can broadcast the group key encoded with the individual key for u. To handle a DELETE(u) operation, the server stops using 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* and 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.

4. 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. [5] 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. But the deletion of user u_j can now be done by simply choosing a new group key and sending it to $M \setminus \{u_j\}$ 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 $M \setminus \{u_j\}$, and thus the delete cost is $\Theta(n)$.

Third, the Key Graph scheme [20,19] 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 in the number of users, then the corresponding add (delete) complexity is at least logarithmic. This suggests we need to argue about sequences and not just about isolated updates.

Canetti et al. [5] 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: $|K(u_i)| \leqslant 3$ 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 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 $2\sqrt{x} - 2 + (n-x)/x$. 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 $j \geqslant 3$ 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 users in X. The simple argument now breaks down. This is why [5] 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.

5. Proof of lower bound

We prove a logarithmic lower bound on key distribution. In Section 5.1 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 5.2 we provide a lower bound on the cost of communications, based on the cost of deleting edges from the graph model. In Section 5.3, we extend our graph model to users without memory, and in Section 5.4 we extend our lower bound to this setting.

5.1. Representing protocol state by a broadcast history graph

We prove our lower bound using graph properties and, so as a first step, we translate the basic group multicast 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 that have a copy of k. Wong et al. [20] defined a *Key Graph* as a bipartite graph with a path from a key k to every user node in *users(k)*. A hierarchically structured representation of this notion 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 it still has the property that for every key k there is a path in the directed graph from node k 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. 2. 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. 2. 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.

We prefer a *broadcast-based history graph* (see Fig. 3) in which nodes correspond 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,

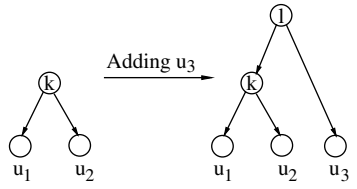


Fig. 2. Simple state graph. The figure shows how the state of a protocol shown on left changes after adding user u_3 in the simple state graph (new state on right). The actual communication is hidden because edges only indicate subset relationships.

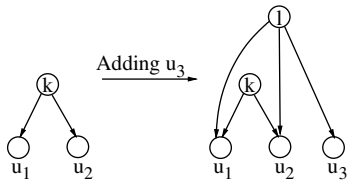


Fig. 3. History-based state graph. How the state of a protocol shown on left changes after adding user u_3. Notice that the actual communication costs incurred so far is recorded in the number of edges in the current graph.

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. 3 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. 2. 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. 3, the root is k at left and l at right.

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. 4. 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. 4 Notice that there is a path between l and u_3 although u_3 does not directly receive a copy of l.

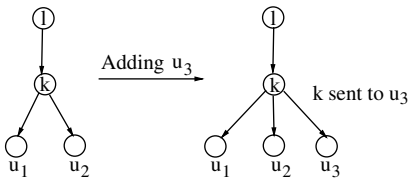


Fig. 4. Sending key k to u_3 creates the graph on the right where l has a path to u_3 but u_3 has no copy of key l.

Suppose that user u_3 has sufficient memory to record broadcast messages received before it 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 5.3.

5.2. 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 broadcast 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 G, and let $w(n)$ denote the minimum value of w_G over all history graphs G with n leaves.

Lemma 5.1. *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 5.2. *The minimum ancestor weight for a broadcast history graph of n leaves satisfies $w(n) \geqslant \lceil 3\log_3 n \rceil$. This is attained by a 2–3 tree.*

Proof. The weight $w(n)$ is an non-decreasing function of n: for any broadcast history graph with $n > 1$ leaves, we can delete one leaf without increasing ancestor weight, so $w(n-1) \leqslant w(n)$.

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 as follows. We start by transforming all nodes of degree 4 and degrees greater than 5 into balanced binary trees. Then, we transform all nodes of degree 5 into a 2–3 tree, with root degree two and one child of degree two and the other of degree three. This replaces a degree-k node by a path of at most $\lceil \log k \rceil$ degree-two nodes, and $k \geqslant 2\lceil \log k \rceil$ for all $k > 3$. Next, merge each node of outdegree one with its parent.

A final transformation moves all the degree-two nodes to the lowest levels of the tree. Repeatedly replace the lowest degree-two node v that is a parent of a degree-three node as follows: if v has two children of degree three, replace them with a degree three parent of degree two children, as in the top half of Fig. 5. Otherwise v has one binary subtree, a portion of which can be replaced by a shorter 3-ary tree as in the lower half of Fig. 5, without increasing the maximum ancestor weight.

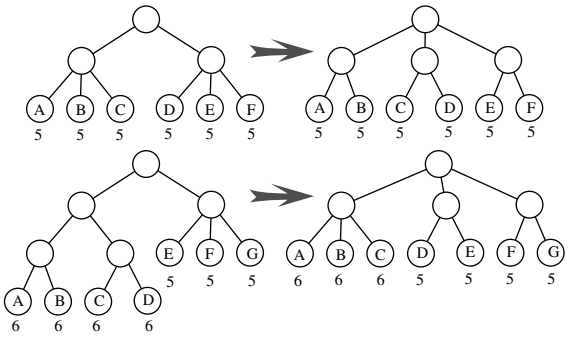


Fig. 5. Example transformations eliminating the lowest degree two parent of a degree three node without increasing weight. Numbers are ancestor weights of the labeled leaves.

Now, consider the maximum number n of leaves in a tree of a given weight $w(n) = k$; since $w(n)$ is an non-decreasing 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, $4 \times 3^{i-1}$, and 2×3^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. 5.) 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 and, therefore, the induction holds. We can summarize by saying that $w(n) \geqslant \lceil 3\log_3 n \rceil$. □

By Lemma 5.1, 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.

It is tempting to infer that the communication cost incurred during the insertion and deletion of a node u is at least as large as the number of edges deleted upon u's deletion. If that were the case then, by our Lemma, either insertion or deletion would have a logarithmic lower bound. Unfortunately, such an inference is flawed, because some of the ancestor node and edges of u could have been created during the insertion of other nodes, not necessarily u. Thus, a bound on the edges deleted after a DELETE() 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.

All we know is that a logarithmic number of edges deleted after every deletion operation must have been created *in some past update operation*. But this bound is sufficient to establish a lower bound on the total cost of a specific sequence of updates, as shown in the following theorem.

Theorem 5.3. *For any secure multicast protocol, there is a sequence of* $2n$ ADD *and* DELETE *operations such that the total communication cost is* $\Theta(n \log n)$ *encrypted messages.*

Proof. Consider a sequence of n insertions of users u_1 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 5.2, the total number of edges deleted must be $\geqslant 3(\log_3 n + \log_3(n-1) + \log_3(n-2) + \ldots \log_3 1)$. Thus the number of edges is at least $3\log_3(n!) = \Theta(n \log n)$. 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 $\Theta(n \log n)$. □

5.3. 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 5.1 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. 4. 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 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. 6 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

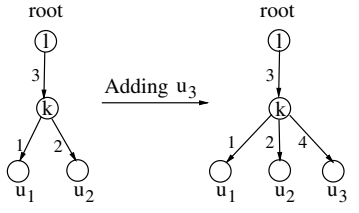


Fig. 6. Adding time stamp information allows us to deduce that u_3 does not have a copy of l because there is no path with non-increasing time stamps from l to u_3.

4, the time stamps increase and this path is not a valid path.

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. 6, the update must first start by deleting nodes k, l, u_1 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 to update the history model is described in Appendix A.

5.4. A lower bound for users without memory

While timestamps do create some new complications, the basic machinery of the ancestor weights in the annotated history graphs works for proving the logarithmic lower bound.

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 therefore define the *valid ancestor weight* for a leaf u_i as follows: Form a subgraph S_i of the annotated history graph that consists of all edges on valid paths to u_i; delete any edge not on some valid path to u_i. The valid ancestor weight for leaf u_i is now defined as the sum of the outdegrees of all vertices in S. The *valid ancestor weight of an annotated history graph* is again the maximum weight of a leaf. We briefly sketch a bound on the minimum valid ancestor weight for a graph of n leaves.

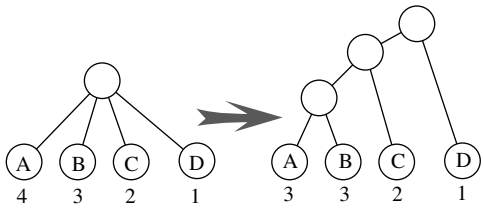


Fig. 7. Transforming a spanning tree with timestamps into a binary tree so that the original valid ancestor weight is bounded from below by node depth.

Lemma 5.4. *The minimum valid ancestor weight for an annotated history graph of n leaves satisfies $w(n) \geqslant \lceil \log_2 n \rceil$.*

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 u with a single child, we contract the earlier of the two edges incident to u to merge that node into its parent or child.

We transform an annotated history tree into a binary tree (Fig. 7) 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 $\lceil \log_2 n \rceil$. □

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

Theorem 5.5. *For any secure multicast protocol, there is a sequence of $2n$ ADD and DELETE operations such that the total communication cost 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 our lemma Lemma 5.4, the total number of edges deleted must be $\geqslant(\log_2 n + \log_2(n-1) + \log_2(n-2) + \cdots \log_2 1)$. 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 $\Theta(n \log n)$. □

6. Conclusion and open questions

We have shown that the logarithmic factor that appears in secure group key maintenance schemes such as RFC 2627 [19] 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. [14], which allows users who leave the group to receive content until a rekeying period. A similar idea is explored by Yang et al. [21], who suggest processing multiple joins and leaves in batches to both reduce the update cost and also to alleviate the out-of-sync problem between rekey messages.

A second approach is to consider *average* instead of *worst* case bounds. For example, in [2], authors show that if each group member can join/leave the group with equal probability then the *average* cost for group re-keying can be constant.

A third approach is to use more powerful ways to build messages out of cryptographic functions (e.g., using nested encryption or other primitives such as pseudorandom generators). Recently, the lower bound has been extended to such more general classes of protocols by Micciancio and Panjwani [12].

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 $n-1$.

Acknowledgments

We wish to thank a reviewer for comments that lead to clarifying the distinction between users with and without memory. Yang et al. [22] independently discovered and proved the same lower bound. We are grateful to Justin Goshi and Richard Ladner for helping us fix an error in the original proof of Lemma 5.2.

Appendix A. A formal model for updating broadcast history graph

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

```
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 u_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 ... 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 labeled 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. A.1. Code for updating annotated broadcast history graph.

current users. (Thus for example the right half of Fig. 6 is an invalid state.) We use CREATEEDGES (Fig. A.1) to represent the creation and broadcasting of new keys by the algorithm. We only require that after the update process 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.

References

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

[2] S. Banerjee, B. Bhattacharjee, Scalable secure group communication over IP multicast, in: Proceedings of Internation Conference on Network Protocols, November 2001; also IEEE Journal on Selected Areas in Communications 20 (8) (2002) 1511–1527.

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

[4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, Multicast security: a taxonomy and efficient authentication, in: Proceedings of INFOCOM'99, March 1999.

[5] R. Canetti, T. Malkin, K. Nissim, Efficient communication-storage tradeoffs for multicast encryption, in: Advances in Cryptology, EUROCRYPT 1999, May 1999.

[6] G. Caronni, M. Waldvogel, D. Sun, B. Plattner, Efficient security for large and dynamic multicast groups, in: Proceedings of Seventh Workshop on Enabling Technologies, (WETICE '98), IEEE Computer Society Press, Silverspring, MD, 1998.

[7] S. Casner, S. Deering, First IETF Internet audiocast, SIGCOMM Computer Communication Review, July 1992.

[8] A. Fiat, M. Naor, Broadcast encryption, in: Advances in Cryptology—CRYPTO'93, 1993.

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

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

[11] M. Luby, J. Staddon, Combinatorial bounds for broadcast encryption, in: Advances in Cryptology—EUROCRYPT'98, 1998.

[12] D. Micciancio, S. Panjwani, Optimal communication complexity of generic multicast key distribution, in: Advances in Cryptology: Proceedings of Eurocrypt 2004, Springer, Berlin, 2004.

[13] S. Mittra, Iolus: a framework for scalable secure multicasting, in: Proceedings of ACM SIGCOMM'97, STW97, September 1997, pp. 277–288.

[14] S. Setia, S. Koussih, S. Jajodia, E. Harder, Kronos: a scalable group rekeying approach for secure multicast, in: IEEE Symposium on Security and Privacy, 2000, pp. 215–228.

[15] C. Shields, J. Garcia-Luna-Aceves, KHIP—a scalable protocol for secure multicast routing, in: Proceedings of ACM SIGCOMM'99, 1999.

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

[17] D. Stinson, T. van Trung, Some new results on key distribution patterns and broadcast encryption, Designs, Codes and Cryptography 14 (3) (1998) 261–279.

[18] R. Thayer, N. Doraswamy, R. Glenn, IP Security Document Road Map, RFC 2411, November 1998.

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

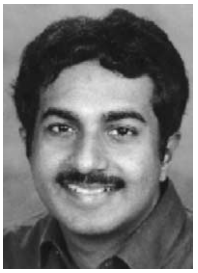
[20] C. Wong, M. Gouda, S. Lam, Secure group communications using Key Graphs, in: Proceedings SIGCOMM 98, September 1998.

[21] Y.R. Yang, X.S. Li, X.B. Zhang, S.S. Lam, Reliable group rekeying: a performance analysis, in: Proceedings SIGCOMM 01, September 2001.

[22] R. Yang, S. Lam, A secure group key management communication lower bound, Technical Report TR-00-24, Department of Computer Sciences, UT Austin, July 2000, revised September 2000.

Jack Snoeyink received Ph.D. in Computer Science from Stanford University in 1990. After a postdoctoral year in Utrecht, he rose through the faculty ranks at the University of British Columbia, and joined the Department of Computer Science at the University of North Carolina at Chapel Hill as a professor in 2000. He works primarily in algorithms for geometric problems, with applications ranging from GIS to molecular biology.

Subhash Suri is a professor in the Department of Computer Science at the University of California, Santa Barbara. His current research interests include algorithms, computational geometry, sensor and mobile networks, algorithmic game theory, and internet computing. He is a member of the ACM and a senior member of the IEEE. He is on the editorial board of the journal Computational Geometry: Theory and Applications, and has acted as guest editor for special issues of journals. He is a reviewer for NSF, NSERC, and numerous journals. He is also a member of many conference program committees, and has served as a consultant for several industrial ventures.



George Varghese received his Ph.D in 1992 from MIT. He is a professor of computer science at UCSD, where he works on efficient protocol implementation and protocol design. Several of the algorithms he has helped develop (e.g., IP Lookups, timing wheels, DRR) have found their way into commercial systems. He is a Fellow of the ACM.