Efficient Tree-Based Revocation in Groups of Low-State Devices

Michael T. Goodrich[1], Jonathan Z. Sun[1], and Roberto Tamassia[2]

Abstract. We study the problem of broadcasting confidential information to a collection of n devices while providing the ability to revoke an arbitrary subset of those devices (and tolerating collusion among the revoked devices). In this paper, we restrict our attention to low-memory devices, that is, devices that can store at most $O(\log n)$ keys. We consider solutions for both zero-state and low-state cases, where such devices are organized in a tree structure T. We allow the group controller to encrypt broadcasts to any subtree of T, even if the tree is based on an multi-way organizational chart or a severely unbalanced multicast tree.

1 Introduction

In the group broadcast problem, we have a group S of n devices and a *group controller* (GC) that periodically broadcasts messages to all the devices over an insecure channel [8]. Such broadcast messages are encrypted so that only valid devices can decrypt them. For example, the messages could be important instructions from headquarters being sent to PDAs carried by employees in a large corporation. We would like to provide for *revocation*, that is, for an arbitrary subset $R \subset S$, we would like to prevent any device in R from decrypting the messages.

We are interested in schemes that work efficiently with low-memory devices, that is, devices that can store at most $O(\log n)$ secret keys. Such a scenario models the likely situation where the devices are small and the secure firmware dedicated to storing keys is smaller still. We refer to this as the *log-key* restriction. We consider two variants of this model.

- A *static* or *zero-state* version: the $O(\log n)$ keys on each device cannot be changed once the device is deployed. For example, memory for the devices could be written into secure firmware at deployment.
- The *dynamic* or *low-state* version: any of the $O(\log n)$ keys on each device can be updated in response to broadcast messages. For example, such devices might have a small tamper-resistant secure cache in which to store and update secret keys.

[1] Dept. of Computer Science, Univ. of California, Irvine, CA 92697-3425. {goodrich, zhengsun}@ics.uci.edu. This work was supported in part by NSF Grants CCR-0312760, CCR-0311720, CCR-0225642, and CCR-0098068.

[2] Dept. of Computer Science, Brown Univ., Providence, RI 02912. rt@cs.brown.edu. This work was supported in part by NSF Grants CCR-0311510, CCR-0098068, and IIS-0324846 and by a research gift from Sun Microsystems.

Organizing Devices Using Trees. The schemes we consider organize the set of n devices in a tree structure, associating each device with a different leaf in the tree. In fact, we consider three possible kinds of trees that the devices can conceptually be organized into.

1. A *balanced d-ary tree*. In this case, the devices are associated with the leaves of a balanced tree where each internal node has a constant d number of children; hence, each is at depth $O(\log n)$. This tree is usually chosen purely for the sake of efficiency, and, in fact, has been the only tree considered in previous related work we are familiar with. For example, it forms the basis of the Logical Key Hierarchy (LKH) scheme [26, 28], the One-way Function Tree (OFT) scheme [21], the Subset-Difference Revocation (SDR) scheme [16], and the Layered Subset Difference (LSD) scheme [10].
2. An *organizational chart*. In this case, the devices are associated with the leaves of a tree that represents an organizational chart, such as that of a corporation or university. For example, internal nodes could correspond to campuses, colleges, and departments. The height of this tree is assumed to be $O(\log n)$ but the number of children of an internal is not assumed to be bounded by a constant. Thus, the straightforward conversion of this tree into an equivalent bounded-degree tree may cause the height to become $\Omega(\log^2 n)$.
3. A *multicast tree*. In this case, the devices are associated with the nodes of a multicast tree rooted at the group controller. The logical structure of this tree could be determined in an ad hoc manner so that no bound is assumed on either the tree height or the degree of internal nodes. Thus, this tree may be quite imbalanced and could in fact have height that is exponentially greater than the number of keys each device can hold.

In using trees, particularly in the latter two cases, we feel it is important to provide the capability to the group controller of encrypting a message so that it may be decrypted only by the devices associated with nodes in a certain subtree. For instance, a sporting event might be broadcast to just a single region, or a directive from headquarters might be intended just for a single division. We call such a broadcast a *subtree broadcast*, which can also be modeled by multiple GCs, each assigned to a different subtree. We continue in this case to assume the network transmits a message to the entire group, even the revoked devices, but it should only be readable by the (unrevoked) devices in the specified subtree when the message is sent in a subtree broadcast. The motivation for organizing devices into trees and allowing for subtree broadcasts is derived from the way many organizations are naturally structured. For example, the ICS Company may have several departments divided into groups, and groups may in turn have divisions located in different cities.

After a secure broadcast system is set up, we need to have the ability to revoke devices to avoid revealing messages beyond the current members. (We also consider the complexities of adding new devices, but the need for revocation is better motivated, since additions will typically be done in large blocks.) Thus, we are interested in the following complexity measures for a set of n devices.

– *Broadcast cost*: the number of messages the group controller (GC) must send in order to reach a subtree containing r revoked devices.
– *Revocation cost*: the number of messages the GC must send in order to revoke a device. Note that this cost is zero in the zero-state case.
– *Insertion cost*: the number of messages the GC must send in order to add a device. Note that this cost parameter does not apply to the zero-state case.

Related Work. Broadcast/multicast encryption was first formally studied by Fiat and Naor [8], for the model where all the device keys are dynamic. Their algorithms satisfy the log-key restriction, however, only if no more than a constant number of revoked devices collude, which is probably not a realistic assumption. Several subsequent approaches have therefore strengthened the collusion resistance for broadcast encryption, and have done so using approaches where the group is represented by a fixed-degree tree with the group controller (GC) being the root and devices (users) being associated with leaves [3–7, 11, 13–15, 23, 24, 26, 28].

Of particular note is the *logical key hierarchy* (LKH) scheme proposed by Wallner *et al.* [26] and by Wong and Lam [28], which achieves $O(1)$ broadcast cost and $O(\log n)$ revocation cost under the log-key restriction (for the dynamic case). The main idea of the LKH scheme is to associate devices with the leaves of a complete binary tree, assign unique secret keys to each node in this tree, and store at each device x the keys stored in the path from x's leaf to the root. Some improvements of this scheme within the same asymptotic bounds are given by Canetti *et al.* [4, 5]. Using Boolean function minimization techniques, Chang *et al.* [6] deal with cumulative multi-user revocations and reduces the space complexity of the GC, i.e., the number of keys stored at the GC, from $O(n)$ to $O(\log n)$. Wong *et al.* [27] generalize the results from binary trees to key graphs. In addition, Sherman and McGrew [21] improve the constant factors of the LKH scheme using a technique they call one-way function trees (OFT), to reduce the size of revocation messages. Naor and Pinkas [17] and Kumar *et al.* [12] also study multi-user revocations withstanding coalitions of colluding users, and Pinkas [18] studies how to restore an off-line user who has missed a sequence of t group modifications with $O(\log t)$ message size. Also of note is work of Rodeh *et al.* [19], who describe how to use AVL trees to keep the LKH tree balanced. Thus, the broadcast encryption problem is well-studied for the case of fully-dynamic keys and devices organized in a complete or balanced k-ary tree (noticing that a k-ary tree can transform to binary with constant times of height increasing). We are not familiar with any previous work that deals with unbalanced trees whose structure must be maintained for the sake of subtree broadcasts, however.

There has also been some interesting recent work on broadcast encryption for zero-state devices (the static case). To begin, we note that several researchers have observed (e.g., see [10]) that the LKH approach can be used in the zero-state model under the log-key restriction to achieve $O(r\log(n/r))$ broadcast cost. (We will review the LKH approach in more detail in the next section.) Naor, Naor, and Lotspiech [16] introduce an alternative approach to LKH, which they call the *subset-difference revocation* (SDR) approach. They show that if devices are allowed to store $O(\log^2 n)$ static keys, then the group controller can send out secure broadcasts using $O(r)$ messages, i.e., the broadcast cost of their approach is $O(r)$. Halevy and Shamir [10] improve the performance of the SDR scheme, using an approach they call layered subset difference (LSD). They show how to reduce the number of keys per device to be $O(\log^{1+\epsilon} n)$ while keeping the broadcast cost $O(r)$. They also show how to further extend their approach to reduce the number of keys per device to be $O(\log n \log\log n)$ while increasing the broadcast cost to be $O(r \log\log n)$. These latter results are obtained using a super-logarithmic number of device keys; hence, they violate the log-key restriction.

Our Results. We provide several new techniques for broadcast encryption under the log-key restriction. We study both the static (zero-state) and dynamic (low-state) versions of this model, and present efficient broadcast encryption schemes for devices organized in tree structures. We study new solutions for balanced trees, organizational charts, and multicast trees. We show in Table 1 the best bounds on the broadcast, insertion and revocation cost for each of the possible combinations of state and tree structure we consider, under the log-key restriction.

		Balanced Tree	**Org. Chart**	**Multicast Tree**
static (zero-state)	broadcast cost	$O(r)$ (new)	$O(r)$ (new)	$O(r \log n)$ (new)
dynamic (low state)	broadcast cost	$O(1)$	$O(1)$	$O(\log n)$
	revocation cost	$O(\log n)$	$O(\log n)$	$O(\log n)$
	insertion cost	$O(\log n)$	$O(\log n)$	$O(\log n)$
		LKH [19, 26, 28]	(new)	(new)

Table 1. Best bounds for broadcast encryption among n devices under the log-key restriction, where each device can store only $O(\log n)$ keys.

So, for example, we are able to match the log-key bound of the static LKH scheme while also achieving the $O(r)$ broadcast encryption complexity of the SDR scheme. Indeed, our scheme for this case, which we call the *stratified subset difference* (SSD) scheme, is the first scheme we are aware of for zero-state devices that simultaneously achieves both of these bounds. Moreover, we are able to match the best bounds for balanced trees, even for unbalanced high-degree organizational charts, which would not be possible using the natural conversion to a binary tree. Instead, we use biased trees [1] to do this conversion. But this approach is nevertheless limited, under the log-key restriction, to cases where the organizational chart has logarithmic height. Thus, for multicast trees, which can be very unbalanced (we even allow for height that is $O(n)$), we must take a different approach. In particular, in these cases, we extend the linking and cutting dynamic trees of Sleator and Tarjan [22] to the context of broadcast encryption, showing how to do subtree broadcasts in this novel context. This implies some surprisingly efficient performance bounds for broadcast encryption in multicast trees, for in severely unbalanced multicast trees the number of ancestors of the leaf associated with some device can be exponentially greater than the number of keys that device is allowed to store.

2 Preliminaries

The LKH Scheme for a Single Group. Let us briefly review the LKH scheme [26, 28], which is well known for key management in single groups. The LKH scheme organizes a group of n devices as a complete binary tree with the GC represented by the root and each user (that is, device) by a leaf, with a key stored at each node. Each device, as a leaf, knows the path from the root to itself and all the keys on this path. The GC, as the root, knows the whole tree and all the keys. (See Figure 1.)

To revoke a device x, the GC updates every key on the path from itself to x so that: (a) x cannot receive any updated key; and (b) any device other than x can receive an updated key if and only if it knows the old value of that key. The key updating is bottom-up, from the parent of x to the root. To distribute the new key at a node v, if v is the parent of x, then the GC encrypts the new key with the current key of the sibling of x; otherwise, GC encrypts the new key with the current keys of the two children of v, respectively. This procedure guarantees (a) and (b). The total number of messages is $O(\log n)$. Broadcasting to a subtree simply involves encrypting a message using the key for the root of that subtree; hence, the broadcast cost is $O(1)$.

In the static case, no updating is allowed. So, the GC must encrypt a broadcast using the root of every maximal subtree containing no revoked devices. Thus, in the static case, LKH has broadcast cost $O(r \log(n/r))$. (Recall that r is the number of revoked devices.) In both the static and dynamic case, however, the number of keys per device remains $O(\log n)$.

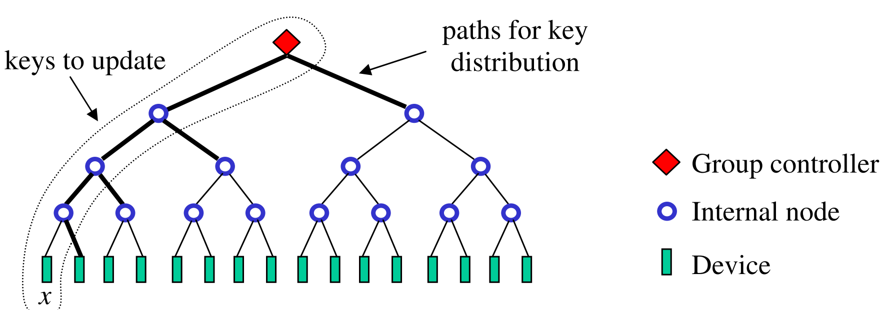

Fig. 1. The LKH scheme for key management in single groups.

Subset Difference Revocation (SDR). The *subset difference revocation* (SDR) approach of Naor, Naor, and Lotspiech [16] is also based on associating all the devices with the leaves of a complete binary tree T. Define a subtree B as the union of all the paths from the root to leaves associated with revoked devices. Some internal nodes in B have one child and some two. Mark each internal node v in B with two children as a "cut vertex" and imagine that we cut out from T the edges from v to its two children. This would leave us with $O(r)$ rooted subtrees, each containing some number of valid devices and one revoked leaf (which may have previously been an internal node). Each such subtree is therefore uniquely identified by its root, v, and its descendent node w that is revoked. The GC associates a secret key with each node v, and defines a label $L_v(w)$, for each node in the subtree, T_v, of T rooted at v. $L_v(v)$ is v's secret key, and for any internal node u in T_v, with left child x and right child y, we define $L_v(x) = f(L_v(u))$ and $L_v(y) = g(L_v(u))$, where f and g are collision-resistant one-way hash functions that maintain the size of input strings. (Here we use the abstract model of f and g; Naor, Naor, and Lotspiech use in [16] a pseudo-random generator G that triples the size of input, and take the left 1/3 and right 1/3 of the output to be the values of f and g.) Each leaf z in T_v stores the values of all the L_v labels of the nodes that are siblings of the path from z to v (that is, not on the path itself, but are siblings of a node on the path). The key used to encode a subtree rooted at v with a revoked node w inside is $L_v(w)$. Note that no descendent of w knows this value and no node outside of T_v can compute this value, which is what makes this a secure scheme. However, this scheme requires each device to hold $O(\log^2 n)$ keys, which violates the log-key restriction.

3 Improved Zero-State Broadcast Encryption

To improve the storage requirements for stateless broadcast encryption, so as to satisfy the log-key restriction, we take a data structuring approach. We begin with the basic approach of the subset difference (SDR) method. Without loss of generality, we assume that we are given a complete binary tree T with n leaves such that each leaf of T is associated with a different user. For any node v in T, let T_v denote the subtree rooted at v. In addition, for any node v and a descendent w of v, we let $T_{v,w}$ denote tree $T_u - T_w$, that is, all the nodes that are descendents of v but not w. Given a set of revoked users, we can use the same approach as SDR to partition T into at most $2r - 1$ subtrees $T_{v,w}$, such that union of all these trees represent the complete set of unrevoked users.

A Linear-Work Solution. As a warm-up for our efficient broadcast encryption scheme, we first describe a scheme that uses $O(\log n)$ keys per device and $O(r)$ messages per broadcast, but requires $O(n)$ work per device to decrypt messages (we will then show how to improve the device work bound keeping the other two asymptotic bounds unchanged).

The main idea is that the GC needs a way of encoding a message so that every leaf node in $T_{v,w}$ can decrypt this message, but not other user (or group of users) can decrypt it. We note as an additional space saving technique, we can name each node in T according to a level-numbering scheme (e.g., see [9]), so that the full structure of any tree $T_{v,w}$ can be completely inferred using just the names of v and w. Moreover, any leaf x in $T_{v,w}$ can determine its relative position in $T_{v,w}$ immediately from its own name, x, and the names of v and w.

Let us focus on a specific subtree T_v, for a node v in T. We define a set of *leftist* labels, $L_v(x)$, and *rightist* labels, $R_v(x)$, for each node of T_v. In particular, let us number the nodes in T_v two ways—first according to a left preorder numbering (which visits left children before right children) and second according to a right preorder numbering (which visits right children before left children) [9]. For a non-root node b in T_v, let a_l denote the predecessor of b in the left preorder numbering of the nodes in T_v. We define $L_v(b)$ to be $f(L_v(a_l))$, where f is a collision-resistant one-way hash function. Likewise, we let a_r denote the predecessor of b in the right preorder numbering of the nodes in T_v. We define $R_v(b)$ to be $g(R_v(a_r))$, where g is a (different) collision-resistant one-way hash function. We initialize these two hash chains by setting $L_v(v)$ and $R_v(v)$ to random seeds known only to the GC.

For each leaf node b in T_v, let c_l and c_r respectively denote the successors of b (if they exist) in the left and right preorder numberings of the nodes in T_v. The keys we store at b for T_v are $L_v(c_l)$ and $R_v(c_r)$. (Note that we specifically do not store $L_v(b)$ nor $R_v(b)$ at b.) For the complete key distribution, we store these two keys for each subtree T_v containing b (there are $\log n$ such subtrees). Given this key distribution, to encrypt a message for the nodes in $T_{v,w}$, a GC encrypts the message twice—once using $L_v(w)$ and once using $R_v(w)$.

Decryption. Let us next consider how a leaf node b in $T_{v,w}$ can decrypt a message sent to this subtree from the GC. Since w is not an ancestor of b, there are two possibilities: either w comes after b in the left preorder numbering of T_v or w comes after b in the right preorder numbering. Since b can determine the complete structure of T_v and b's relative position with w in this subtree from the names of v, b, and w, it can implicitly represent $T_{v,w}$ and know which of these two cases apply. So suppose the first case applies (as the second case is symmetric with the first). In this case, b starts with the label $L_v(c_l)$ it stores, where c_l is b's successor in the left preorder numbering of T_v. It then continues a left preorder traversal of T_v (which it can perform implicitly if memory is tight) until it reaches w. With each new node b encounters in this traversal, b makes another application of the one-way function f, computing the L_v labels of each visited node. Thus, when b visits w in this traversal, it will have computed $L_v(w)$ and can then decrypt the message. This computation takes at most $|T_{v,w}|$ hash function computations.

Security. Let us next consider the security of this scheme. First, observe that any node outside of T_v has no information that can be used to help decode a message for the nodes in some tree $T_{v,w}$, since $L_v(v)$ and $R_v(v)$ are chosen as random seeds and nodes outside of T_v receive no function of $L_v(v)$ or $R_v(v)$. So the security risk that remains is that leaf descendents of w might be able to decrypt a message sent to the nodes in $T_{v,w}$. Let D_w denote the set of leaf descendents of w. For each node b in D_w, with successors c_l and c_r in the two preorder numberings, we store $L_v(c_l)$ and $R_v(c_r)$ at b. But none of these values for the nodes in D_w are useful for computing $L_v(w)$ or $R_v(w)$, without inverting a one-way function, since, in any preorder traversal, all the ancestors of a node are visited before the node is visited.

Thus, we have a key distribution strategy for the zero-state case that uses $O(\log n)$ keys per device and $O(r)$ messages per broadcast, albeit with work at each device that could be $O(n)$. In the remainder of this section, we describe how we can reduce this work bound while keeping the other asymptotic bounds unchanged.

The Stratified Subset Difference (SSD) Method. Given a constant k, we can decrease the work per device to be $O(n^{1/k})$, while increasing the space and message bounds by at most a factor of k, which should be a good trade-off in most applications. For example, when n is less than one trillion, $n^{1/8}$ is less than $\log n$. The method involves a stratified version of the scheme described above, giving rise to a scheme we call the *stratified subset difference* (*SSD*) method.

We begin by marking each node at a depth that is a multiple of $\lceil (\log n)/k \rceil$ as "red;" the other nodes are colored "blue." (See Figure 2.) Imagine further that we partition the tree T along the red nodes, subdividing T into maximal trees whose root and leaves are red and whose internal nodes are blue. Call each such tree a *blue tree* (even though its root and leaves are red). We then apply the method described above in each blue tree, as follows. For each leaf b in T, let $b_1, \cdots, b_k$ be the red ancestors of b, in top-down order. For $i = 1, \cdots k$, let T_i be the blue tree rooted at b_i and note that b_{i+1} is a leaf of T_i.

We store at node b labels $L_{b_i}(c_l)$ and $R_{b_i}(c_r)$ ($i = 1, \cdots k$ in T), where c_l and c_r are the left and right preorder successors of b_{i+1} in T_i, respectively. Storing these labels increases the space per device by a factor of k.

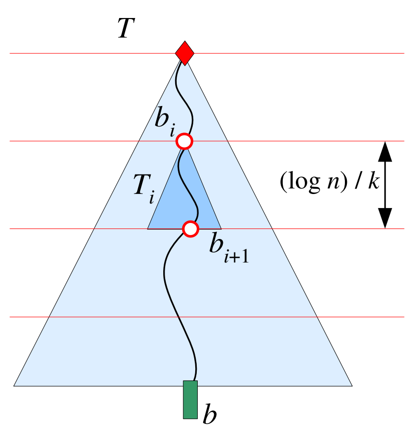

Fig. 2. Illustration of the stratified subset difference (SSD) scheme.

To encrypt a message, the GC first performs the subdivision of T into the subtrees $T_{v,w}$ as before. Then, the GC further partitions each tree $T_{v,w}$ at the red levels, and encodes the broadcast message, using the previously described scheme, for each blue subtree rooted at a node on the path from v to w. This increases the broadcast size by at most a factor of k, but now the work needed by each device is reduced to computing the L or R labels in a blue tree, which has size at most $n^{1/k}$. Thus, the work per device is reduced to $O(n^{1/k})$ in this SSD scheme.

Theorem 1. *Given a balanced tree T with n devices, for zero-state broadcast encryption, the stratified subset difference (SSD) scheme for T uses $O(\log n)$ keys per device and has $O(r)$ broadcast cost, where r is the number of revoked devices in the subtree receiving the broadcast. The work per device can be made to be $O(n^{1/k})$ for any fixed constant k.*

Moreover, as we have noted, the security of this scheme is as strong as that for SDR and LKH, i.e., it is resilient to collusions of any set of revoked devices.

4 A Biased Tree Scheme for an Organizational Chart

We recall that in the organizational chart structure for n devices, we have a hierarchical partition of the devices induced by a tree T of $k = O(\log n)$ height but with unbounded branches at each internal node. Namely, the leaves of T are associated with the devices and an internal node v of T represents the group (set) of devices associated with the leaves of the subtree rooted at v. Thus, sibling nodes of T are associated with disjoint groups and each device belongs to a unique sequence of $O(\log n)$ groups whose nodes are on the path from the device's leaf to the root of T. Without loss of generality, we assume that an internal node of T has either all internal children (subgroups) or all external children (devices), and its group is called an interior group or exterior group accordingly. We consider four types of update operations: *insertion* and *deletion* (revocation) of a device or of an empty group. After each modification, we want to maintain both forward and backward security.

Biased Trees. Biased trees, introduced by Bent *et al.* [1], are trees balanced by the weights of leaves (typically set as access frequencies). There are two versions of biased trees: locally biased and globally biased. We denote by $p(x)$, $l(x)$ and $r(x)$ the parent, left child and right child of a node x of a tree, and we use these denotations cumulatively. E.g., $lpp(x)$ is the left child of the grandparent of x. The following definitions are taken from [1].

A *biased search tree* is a full binary search tree such that each node x has a weight $w(x)$ and a rank $s(x)$. The weight of a leaf is initially assigned, and the weight of an internal node is the sum of the weights of its children. The rank $s(x)$ of a node x is a positive integer such that

1. $s(x) = \lfloor \log w(x) \rfloor$ if x is a leaf.

2. $s(x) \leq s(p(x)) - 1$ if x is a leaf.
3. $s(x) \leq s(p(x))$ and $s(x) \leq s(pp(x)) - 1$.

A *locally biased* search tree has the following additional property:

Local bias. For any x with $s(x) \leq s(p(x)) - 2$,

1. if $x = lp(x)$, then either $rp(x)$ or $lrp(x)$ is a leaf with rank $s(x) - 1$; if $x = rp(x)$, then either $lp(x)$ or $rlp(x)$ is a leaf with rank $s(x) - 1$; and
2. if $x = lp(x)$, $p(x) = rpp(x)$ and $s(p(x)) = s(pp(x))$, then either $lpp(x)$ or $rlpp(x)$ is a leaf with rank $s(x) - 1$; if $x = rp(x)$, $p(x) = lpp(x)$ and $s(p(x)) = s(pp(x))$, then either $rpp(x)$ or $lrpp(x)$ is a leaf with rank $s(x) - 1$.

A *globally biased* search tree has the following additional property:

Global bias. For any x with $s(x) \leq s(p(x)) - 2$, both of the two neighboring leaves of x, i.e., the right-most leaf on the left and the left-most leaf on the right, have rank at least $s(x) - 1$.

Group Hierarchies and Biased Trees. Given an organizational chart T that represents a group hierarchy, we have to convert T to a binary tree before applying any encryption scheme for key management. Without loss of generality, we convert T to a binary tree B_T that preserves the original group hierarchy. Each internal node of T, representing a group G_i, becomes a special internal node in B_T that still represents G_i and accommodates a GC. Additional internal nodes are added between G_i and its children in T (i.e., subgroups or devices) for the purpose of binarization. As result,node G_i plus all its children in T and the paths between them in B_T form a binary subtree B_i in B_T with G_i being the root and each of its children in T being a leaf. Note that, without special care, B_T is likely to have super-logarithm height and balancing such a tree using standard techniques would destroy the group hierarchy.

Given a group hierarchy tree T, we assign a unit weight to each leaf and calculate the weights of other nodes in T accordingly, i.e., the weight of each internal node x is the number of devices in the subtree of T rooted at x. We replace each node x with a biased binary tree having the children of x as its leaves (using the weights of these nodes for the biasing). Thus, each subtree B_i representing a group G_i rooted at a node x in T can be initialized into a biased tree without affecting the structure of group hierarchy. Since $w(G_i)$ for each G_i is an invariant, i.e., the weights of the root and leaves in every B_i are invariant, the initialization is well defined and can be done in each B_i independently. That is, combining all the biased B_i's into B_T will not change the structure of the original hierarchy represented by T. (See Figure 3)

Key assignment. After initializing the biased B_i's, we still assign a key to each node of B_T as in the LKH, and inform the keys to devices and GC's by the following security properties:

1. each device x knows all but only the keys on the path from G_0 to itself.
2. the GC of each G_i knows all but only the keys of G_i's descendants in B_T and those on the path from G_0 to G_i.

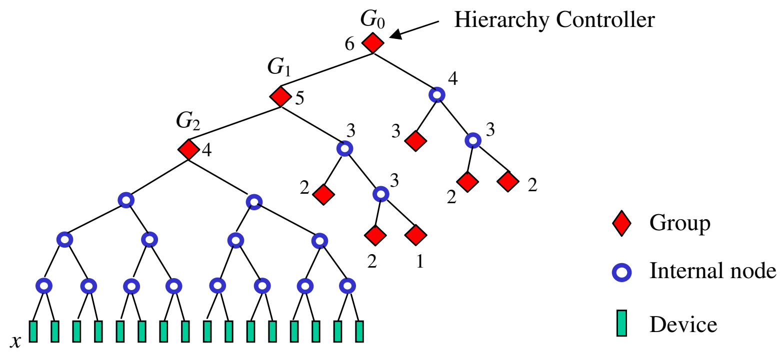

Fig. 3. Binary tree B_T consisting of biased trees B_0, B_1 and B_2. The ranks of the nodes in B_0 and B_1 are shown.

Broadcast and multicast. Using the above security properties and appropriate signature or authentication mechanism [2, 4, 20, 25], the GC of each G_i can send a message securely with one key encryption to G_i or any subgroup or super-group of G_i, without any ambiguity.

Key update and tree rebalance. As in the LKH scheme, keys should be updated after each *insertion* or *deletion* (revocation) of a device or group so that the security properties 1 and 2 are maintained. Moreover, we should also rebalance B_T to preserve the bias properties in each B_i. Assume that we can insert a leaf, delete a leaf, or update the weight of a leaf in B_i (by $insert(x)$, $delete(x)$ and $reweight(x)$, respectively) while preserving both the security and bias properties. Then inserting or deleting a device $x \in G_k \subset G_{k-1} \subset \cdots \subset G_0$ can be done in three steps:

1. insert or delete a leaf in the exterior tree B_k;
2. update the weights $w(G_k), w(G_{k-1}), \cdots, w(G_1)$ in the interior trees B_{k-1}, $B_{k-2}, \cdots, B_0$ accordingly; and
3. update the keys on the path from x to G_0 bottom-up, as in the LKH scheme.

To insert or delete a group $G_{k+1} \subset G_k \subset \cdots \subset G_0$ is a similar process except starting with an insertion or deletion in an interior B_k. Therefore *insert*, *delete* and *reweight* in each B_i suffice all our hierarchy modifications in B_T. Such operations preserving the bias properties were already given and analyzed in [1], we now describe how to modify them to preserve the security properties, too.

Recall that the biased tree operations, including *insert*, *delete* and *reweight*, recursively call an operation *tilt* as the only subroutine to rebalance the biased tree structure [1]. Operation *tilt* performs a single rotation associated with rank modification. Since a node loses descendants during a rotation if it is rotated down and losing descendants is the only chance of key leaks in the LKH scheme. To maintain the security properties 1 and 2 after any rotation in B_i, it is necessary and sufficient to update the key at the node rotated down. Observing that updating a single key and distributing the result of a rotation are both easy in our scheme, we can replace the *tilt* in [1] with our *secure-tilt* which preserves the security properties 1 and 2.We give a detailed description of *secure-tilt-left* in Figure 4. Operation *secure-tilt-right* is analogous. Using *secure-tilt* as the subroutine in biased tree operations, the scheme is as secure as LKH.

Algorithm *secure-tilt-left*(x)
  if $s(l(x)) = s(x) = s(r(x))$ **then**
    $s(x) \leftarrow s(x) + 1$
  else if $s(l(x)) < s(x)$ and $s(r(x)) = s(x)$ **then**
    let $x, l(x), r(x), lr(x), rr(x)$ be A, B, C, D, E.
    $p(C) \leftarrow p(A)$, $l(C) \leftarrow A$ and $r(A) \leftarrow D$. {left rotation at x}
    update *key*(A)
    distribute *key*(A) and *key*(C) to their descendants
    $x \leftarrow C$
  end if
  return x

Fig. 4. The algorithm for operation *secure-tilt-left*(x).

Efficiency of The Scheme. The *insert*, *delete* and *reweight* operations in biased trees are implemented as follows: *join* and *split* are the two basic biased tree operations. *join*(x, y) has global and local versions, which will merge two global or local biased trees with roots x and y and return the root of the resulting tree, and both versions work by recursively calling *secure-tilt*. *split*(T, x) will split T into two biased trees T_1 and T_2, each containing all the leaves of T with their binary search keys less than x and greater than x, respectively. *split* calls *local-join* as a subroutine and is applicable to both local and global biased trees.

Other operations are based on *join* and *split*: operation *insert*(x) splits T by x and then joins T_1, x and T_2 together; operation *delete*(x) splits T by x and then joins T_1 and T_2 back ignoring x; and operation *reweight*(x) splits T by x, updates the weight of x, and then joins T_1, x and T_2 back into T.

The correctness and efficiency of our hierarchy modifications in B_T follow those of biased tree operations. Notice that our *secure-tilt* takes constant message size as well as the constant-time *tilt* in [1], all time bounds in [1] also hold as bounds of message size in our scheme.

This gives us the following.

Theorem 2. *Given an organizational chart tree T with height k and n devices, under the log-key restriction, the dynamic biased binary tree scheme for T has has $O(1)$ broadcast cost and $O(k + \log n)$ revocation and insertion cost.*

Proof. We show how to access a device $x \in G_k \subset G_{k-1} \subset \cdots \subset G_0$ from G_0. The analysis of other operations is similar. Since the root of B_i is a leaf of B_{i-1}, and each biased tree B_i, $i = 0, 1, \cdots, k$, has the ideal access time, the time to access x from G_0 is

$$O\left(\left\lceil \log \frac{w(G_0)}{w(G_1)} \right\rceil + \cdots + \left\lceil \log \frac{w(G_{k-1})}{w(G_k)} \right\rceil + \left\lceil \log \frac{w(G_k)}{w(x)} \right\rceil\right) = O(\log n + k). \qquad \square$$

Thus, we satisfy the log-key restriction for any organizational chart with $k = O(\log n)$ height. We also note that applying our SSD approach to a static application of the techniques developed in this section results in a scheme using $O(\log n)$ keys per device and $O(r)$ messages per broadcast for an organization chart with height $O(\log n)$.

5 A Dynamic Tree Scheme for a Multicast Tree

Let us next consider the multicast tree structure, which, for the sake of broadcast encryption, is similar to the organizational chart, except that the height of a multicast tree can be much larger than logarithmic (we even allow for linear height). For a multicast tree T with n devices and m groups, we give a scheme with $O(\log m)$ broadcast cost and $O(\log n)$ update cost, irrespectively of the depth of T.

Dynamic Trees. Dynamic trees were first studied by Sleator and Tarjan [22] and used for various tree queries and network flow problems. The key idea is to partition a highly unbalanced tree into paths and associate a biased tree structure, which is in some sense balanced, to each path. Thus any node in the tree can be accessed and any update to the tree can be done in $O(\log n)$ time through the associated structure, regardless the depth of node or the height of tree. The dynamic tree used in our scheme is specified by taking the partition by weight (size) approach and not having cost on each edge. The following definition refers to this specification.

A *dynamic tree* T is a weighted binary search tree where the weight $w_T(x)$ is initially assigned if x is a leaf, or $w_T(x) = w_T(l(x)) + w_T(r(x))$ if x is an internal node. The edges of T are partitioned into solid and dashed edges so that each node links with its heavier child by a solid edge and with the lighter child by a dashed edge. Thus T is partitioned into solid paths P_j's linked by dashed edges. We denote by $h(P_j)$ the deepest node in P_j and $t(P_j)$ the upper-most one[1]. Then the edge between any $t(P_j)$ and its parent must be dashed, and vice versa. For $O(\log n)$ operations, each solid path P_j is further organized as a global biased tree, denoted by $B(P_j)$, so that the nodes from $h(P_j)$ to $t(P_j)$ become leaves of $B(P_j)$ from left to right, and the weight of a leaf x in $B(P_j)$ is assigned as $w_{B(P_j)}(x) = w_T(y)$ where y is the dashed child of x in T. Then T consists of these $B(P_j)$'s by linking the root of each $B(P_j)$ with the parent of $t(P_j)$, unless $t(P_j)$ is the root of T. (See Figure 5.) To show that such structure of T is well defined, let the root of $B(P_j)$ be x and the parent of $t(P_j)$ be $y \in P_{j-1}$, then we have that $w_{B(P_j)}(x) = w_T(t(P_j)) = w_{B(P_{j-1})}(y)$. Thus, x can replace $t(P_j)$ as a child of y.

[1] $h(P_j)$ must be a leaf of T by the "partition by weight" approach.

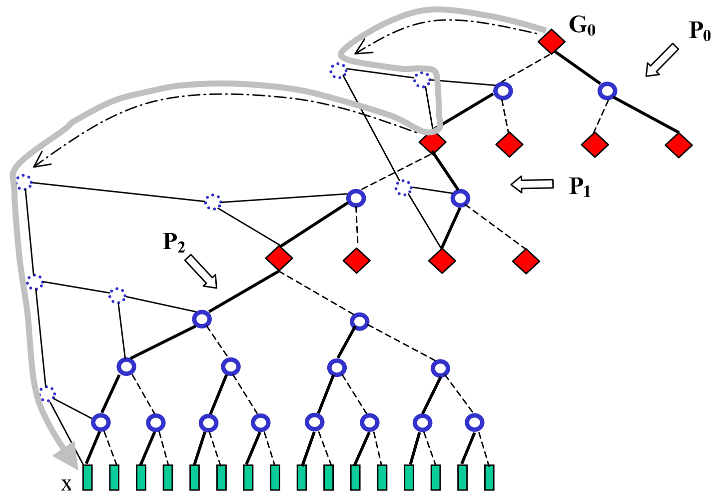

Fig. 5. Partition of tree B_T and the accessing path to x.

Group Hierarchies and Dynamic Trees. We convert a multicast tree T to a binary tree B_T that preserves the group hierarchy in T as same as in the biased tree scheme. Instead of using a biased tree, we simply use a complete binary tree for each B_i, then assign a unit weight $w_T(x) = 1$ to each device and partition B_T into a dynamic tree as above. A key is assigned to each node of each $B(P_j)$. Since the root of $B(P_j)$ becomes child of a leaf of $B(P_{j-1})$, each device becomes a descendant of a unique string of biased trees of paths $B(P_j), B(P_{j-1}), \cdots, B(P_0)$. The way a device is accessed is not through the real path in B_T but through the path in the string of $B(P_j)$'s. (See Figure 5.)

Broadcast and Multicast. Broadcast in a group G_i becomes a little more complicate because, although device x is a descendant of G_i in T, G_i may not be on the accessing path from G_0 to x. However, if $G_i \in P_j$, then the accessing path to any descendant of G_i must pass a node in the prefix of P_j from $h(P_j)$ to G_i. So, to broadcast in G_i, it is sufficient to encrypt the message by the keys in $B(P_j)$ that cover this prefix of P_i. In the full version, we show that, with the dynamic tree scheme, it takes $O(\log |P_j|)$ encryptions to broadcast a message in any group $G_i \in P_j$, either in worst case or in average.

Key Updates. We follow the dynamic tree operations in [22] to modify the hierarchy, and update the keys in the accessing path of the updated item as in the LKH scheme. Dynamic tree operations dynamically change the solid path partition to guarantee the $O(\log n)$ running time, and such change is carried out by the biased tree operations among $B(P_j)$'s. Therefore, operation *secure-tilt* preserves the security properties along any accessing path. The dynamic tree operations we use are as follows:

- *splice*(P_j): extend P_j by converting the edge from $t(P_j)$ to its parent solid, and the edge between $sibling(t(P_i))$ and its parent dashed.
- *slice*(P_j): Let (x, y) be the upper most edge in P_j such that y is not the heavier child of x, if there exist such edges in P_j. Then cut P_j by converting (x, y) into dashed and $(x, sibling(y))$ into solid.
- *expose*(x): make the path from x to G_0 (the real path in B_T) into a single solid path by a series of *splices*.
- *conceal*(P_j): convert every edge in P_j who does not link to a heavier child of parent into dashed by a series of *slices*.
- *link*(x, y): combine two dynamic trees by making y the parent of x, where x is the root of the first tree and y is a node in the second.
- *cut*(x): divide a dynamic tree into two by deleting the edge between x and $p(x)$.

Inserting or deleting a device or a group corresponds to a *link* or *cut* operation, respectively. Such dynamic tree operation take $O(\log n)$ time and can be reduced to a series of *join* and *split* operation on biased trees. The algorithmic template for a dynamic tree operation is the *expose-and-conceal* strategy, described as follows:

1. perform *expose*(x) on a node x;
2. if the above *expose* operation violates the "partition by weight" property, restore the property by executing *conceal*(P_j) on the appropriate path P_j.

Since all the dynamic tree operations reduce to a series of biased tree operations, operation *secure-tilt* is still the only subroutine that adjusts the partition of $T(P_j)$'s. Notice that the structure B_T is never adjusted, but the accessing path to each device x are adjusted through operations. From [22], we know that, with partition by weight and representing the solid paths as global biased trees, any dynamic tree operation takes $O(\log n)$ time. Since a hierarchy modification consists of a dynamic tree operation plus updating the keys in an access path, which is also of length $O(\log n)$, the efficiency of key updating for hierarchy modifications follows.

Theorem 3. *Given a multicast tree T with n devices, under the log-key restriction, structured in m groups, the dynamic tree scheme for T has $O(\log m)$ broadcast cost and $O(\log n)$ revocation and insertion cost.*

A zero-state version can also be developed, which uses the biased trees and broadcast scheme to send messages to the unrevoked leaves in a multicast tree T using $O(r \log n)$ broadcasts for devices storing $O(\log n)$ keys each, where r is the number of revoked devices.

References

1. S. W. Bent, D. D. Sleator, and R. E. Tarjan. Biased search trees. *SIAM J. Comput.*, 14(3):545–568, Aug. 1985.
2. D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *Proc. EUROCRYPT 2001, LNCS 2045*, pages 437–452, May 2001.
3. B. Briscoe. Marks: Zero side effect multicast key management using arbitrarily revealed key sequences. In *Proc. of First International Workshop on Networked Group Communication(NCGC'99)*, 1999.
4. R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proc. INFOCOM '99*, volume 2, pages 708–716, New York, Mar. 1999.
5. R. Canetti, T. Malkin, and K. Nissim. Efficient communication — storage tradeoffs for multicast encryption. In *Advances in cryptology (EUROCRYPT'99), LNCS 1592*, pages 459–474, 1999.
6. I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key management for secure Internet multicast using boolean function minimization techniques. In *Proc. IEEE INFOCOM*, volume 2, pages 689–698, 1999.
7. G. D. Crescenzo and O. Kornievskaia. Efficient kerberized multicast in a practical distributed setting. In *4th International Conference Information Security (ISC'01), LNCS 2200*, pages 27–45, Oct. 2001.
8. A. Fiat and M. Naor. Broadcast encryption. In *Advances in Cryptology - CRYPTO'93*, pages 480–491. LNCS 773, 1994.
9. M. T. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis and Internet Examples*. John Wiley & Sons, New York, NY, 2002.
10. D. Halevy and A. Shamir. The LSD broadcast encryption scheme. In *Advances in Cryptology (CRYPTO 2002)*, volume 2442 of *LNCS*, pages 47–60. Springer-Verlag, 2002.
11. E. Jung, A. X. Liu, and M. G. Gouda. Key bundles and parcels: Secure communication in many groups. In *Proceedings of the 5th International Workshop on Networked Group Communications (NGC-03), LNCS 2816*, pages 119–130, Munich, Germany, September 2003.
12. R. Kumar, R. Rajagopalan, and A. Sahai. Goding constructions for blacklitting problems without computational assumptions. In *Advances in cryptology (CRYPTO'99), LNCS 1666*, pages 609–623, 1999.
13. D. A. McGrew and T. Sherman. Key establishment in large dynamic groups using one-way function trees. Technical Report 0755, TIS Labs at Network Associates Inc., Glenwood, MD, May 1998.
14. D. Micciancio and S. Panjwani. Optimal communication complexity of generic multicast key distribution. In *EUROCRYPT 2004*, pages 153–170, 2004.
15. M. J. Mihajevic. Key management schemes for stateless receivers based on time varying heterogeneous logical key hierarchy. In *ASIACRYPT 2003, LNCS 2894*, pages 137–154, 2003.
16. D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO'01*, volume 2139 of *LNCS*, pages 41–62. Springer-Verlag, 2001.
17. M. Naor and B. Pinkas. Efficient trace and revoke schemes. In *Proc. Financial Crypto 2000*, Feb. 2000.
18. B. Pinkas. Efficient state updates for key management. In *Proc. ACM Workshop on Security and Privacy in Digital Rights Management*, 2001.

19. O. Rodeh, K. P. Birman, and D. Dolev. Using AVL trees for fault tolerant group key management. *International Journal on Information Security*, pages 84–99, 2001.
20. B. Schneier. *Applied Cryptography, 2nd Ed.* John Wiley - Sons, 1996.
21. A. T. Sherman and D. A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Trans. Software Engineering*, 29(5):444–458, 2003.
22. D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Computer and System Sciences*, 26:362–391, 1983.
23. J. Snoeyink, S. Suri, and G. Varghese. A lower bound for multicast key distribution. In *IEEE INFOCOM 2001*, volume 1, pages 422–431, 2001.
24. R. Tamassia and N. Triandopoulos. Computational bounds on hierarchical data processing with applications to information security. Technical report, Center for Geometric Computing, Brown University, 2004.
25. H. F. Tipton and M. Krause, editors. *Information Security Management Handbook, 4th Ed.* Auerbach, 1999.
26. D. M. Wallner, E. G. Harder, and R. C. Agee. Key management for multicast: issues and architecture. In *internet draft draft-waller-key-arch-01.txt*, Sep. 1998.
27. C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. In *Proc. ACM SIGCOMM'98*, volume 28, pages 68–79, 1998.
28. C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking*, 7:502–513, 1999.