Key Establishment in Large Dynamic Groups Using One-Way Function Trees

Alan T. Sherman and David A. McGrew, Member, IEEE

Abstract—We present, implement, and analyze a new scalable centralized algorithm, called OFT, for establishing shared cryptographic keys in large, dynamically changing groups. Our algorithm is based on a novel application of one-way function trees. In comparison with the top-down logical key hierarchy (LKH) method of Wallner et al., our bottom-up algorithm approximately halves the number of bits that need to be broadcast to members in order to rekey after a member is added or evicted. The number of keys stored by group members, the number of keys broadcast to the group when new members are added or evicted, and the computational efforts of group members, are logarithmic in the number of group members. Among the hierarchical methods, OFT is the first to achieve an approximate halving in broadcast length, an idea on which subsequent algorithms have built. Our algorithm provides complete forward and backward security: Newly admitted group members cannot read previous messages, and evicted members cannot read future messages, even with collusion by arbitrarily many evicted members. In addition, and unlike LKH, our algorithm has the option of being member contributory in that members can be allowed to contribute entropy to the group key. Running on a Pentium II, our prototype has handled groups with up to 10 million members. This algorithm offers a new scalable method for establishing group session keys for secure large-group applications such as broadcast encryption, electronic conferences, multicast sessions, and military command and control.

Index Terms—Broadcast encryption, conference keying, cryptography, cryptographic protocols, Dynamic Cryptographic Context Management (DCCM) Project, group keying, key agreement, key establishment, key management, logical key hierarchy (LKH), one-way functions, one-way function chain (OFC), one-way function tree (OFT), secure conferences, secure group applications.

1 INTRODUCTION

Efficiently managing cryptographic keys for large, dynamically changing groups is a difficult problem. Every time a member is evicted from a group, the group key must change; it may also be required to change when new members are added. The members of the group must be able to compute a new key efficiently, while arbitrary coalitions of evicted members must not be able to obtain it. Subject to these security constraints, we also seek to minimize the communication, computation, and storage costs of rekeying. The difficulty of this problem stems from two sources. First, to rekey everyone in a large group quickly, it is necessary to avoid the inefficient strategy of sending a separate rekeying message to each member. Second, as evidenced by flaws in some previously proposed methods, it is challenging to achieve security against collusion attacks.

Real-time applications need very fast rekeying so that changes in group membership are not disruptive. Such applications include secure audio and visual broadcasts, pay TV, secure conferencing, controlling access to broadcast satellite services, and military command and control. To deal with large group sizes (e.g., 100,000 members and larger), we seek solutions whose rekeying operations “scale” well in the sense that time, space, and broadcast requirements of the method grow at most logarithmically in the group size. Key management for these applications should be able to take advantage of efficient broadcast channels, such as land or satellite radio broadcast, cable TV, and Internet multicast. Securing whole-earth satellite broadcasts is an especially attractive application for OFT.

We present and analyze a new practical algorithm called One-Way Function Tree (OFT) for establishing shared keys in large, dynamic groups. Our algorithm, which is based on a novel application of one-way function trees, scales logarithmically in group size. Thus, for a group with 10 million members, an evict operation requires only approximately 23 message units (each proportional to the size of a cryptographic key) to be broadcast. In comparison with previously published methods—including the top-down Logical Key Hierarchy (LKH) of Wallner et al. [57] and Harney and Harder [29] in particular—our bottom-up algorithm approximately halves the required broadcast size. Developed in 1997, OFT is the first of the hierarchical methods to achieve such an approximate halving of broadcast size—an idea on which subsequent algorithms have built. Also, in contrast with LKH, our OFT method has the option of allowing group members to contribute entropy to the group key.

OFT achieves perfect forward security and perfect backward security. Forward security means that evicted members cannot learn the new group key. Backward security means that newly added members cannot learn previous group keys. Here, the adjective “perfect” adds the constraint that these properties must remain true even against
arbitrarily many colluding attackers. Thus, perfect forward security means that evicted members cannot determine any future group key, even with collusion by arbitrarily many evictees. Similarly, perfect backward security means that newly added members cannot determine any previous group key, even with collusion by arbitrarily many new members. It is not interesting to consider collusion attacks that include current members because every group member knows the group key.

We present our algorithm in a general setting that permits both multicast and unicast. When applied to typical multicast environments that allow multicasts only, each unicast operation would be implemented as a multicast.

The main contribution of this paper is our invention, description, and analysis of OFT. We hope that this paper will also be useful for our review of previous and subsequent related work, our implementation experience with OFT, and our broader discussion of group keying.

The rest of this paper is organized in seven sections, which describe the OFT algorithm, its computational properties, and our implementations of it. Section 2 briefly reviews previous and subsequent approaches to group key establishment. Section 3 explains our hierarchical model for group keying and provides a useful common framework and terminology for describing centralized hierarchical methods. Section 4 describes our proposed method, and Section 5 gives an enhanced version with constant-time add-member operation. Section 6 analyzes the algorithm’s performance. Section 7 discusses implementation issues based on our experience with our prototype implementations. Section 8 summarizes our conclusions.

The related full-length technical report by Sherman and Mcgrew [50] includes additional figures, pseudocode, and discussions of security and how to manage logical subgroups. A separate companion paper by Sherman [51] studies the security of OFT and related algorithms. This companion paper includes security definitions, proofs of security by reduction, and an analysis of the loss of effective entropy in the group key due to iterated functions.

2 Previous and Subsequent Work

A variety of group keying solutions have been proposed, including methods based on a Simple Key Distribution Center (SKDC), information theoretic approaches, Group Diffie-Hellman (GDH), hybrid approaches that trade off information theoretic security against storage requirements, and recent centralized hierarchical methods including the Logical Key Hierarchy (LKH). In this section, we briefly review some of these methods, including ones that were developed before or after OFT.

2.1 Linear, Information-Theoretic, and Distributed Methods

The simplest solution is Simple Key Distribution Center (SKDC), in which a group manager shares a secret key with each group member and sequentially uses each member’s key to communicate the secret group key to that member. Each time that a member is added to (or evicted from) a group with \( n \) members, the group manager must perform \( n \) encryptions and transmit \( n \) keys. This approach is attractive for its simplicity—at least for relatively small groups (e.g., see [28], [30], [31], [32]). A limitation of SKDC is apparent in the Spread System [2], which cannot handle groups of more than a few thousand members. By contrast, we are especially interested in handling groups of size 100,000 or more.

Information theoretic approaches must satisfy the memory lower bounds of Blundo et al. [11] and use storage exponential in group size. All of the previously published hybrid approaches that achieve perfect forward security also scale at least linearly in group size. For example, for resilience against \( n/2 \) colluders, the hybrid method of Fiat and Naor [22] requires a broadcast length of \( O(n^2 \log^2 (n \log n)) \), where \( n \) is the group size.

Although Group Diffie-Hellman (GDH) methods [54], [55], [14], [13], [3] offer distributed functionality, which might be attractive for some applications, many such proposals have suffered from a linear number of expensive public-key operations. For example, the recent STR method of Kim et al. [34] has a reduced number of rounds but requires about \( 3n/2 \) serial exponentiations. When the network is a tree, however, GDH methods can require only a logarithmic number of operations [14], [1]. In particular, recent work of Tsudik et al. [35], applies hierarchical ideas to a distributed environment. Nevertheless, distributed computation is not appropriate for all applications. Furthermore, the basic unit of cost for all GDH methods includes public-key operations, which are slow in software relative to symmetric encryption, one-way function, or pseudorandom function operations.

2.2 Centralized Hierarchical Methods

For large groups that permit a centralized approach, the leading candidates are the centralized hierarchical methods, which scale logarithmically in group size. Recently, three such hierarchical methods have been proposed that do not require trusted internal nodes: the Logical Key Hierarchy (LKH) of Wallner et al. [57] and Harney and Harder [29], the One-Way Function Tree (OFT) method of Sherman and Mcgrew [50], [38], and Balenson et al. [5], [6], and the One-Way Function Chain (OFC) of Canetti et al. [15], Section 4.2.

LKH is a top-down method in that it “pushes” new group keys down the key tree; OFT and OFC are bottom-up methods in that they derive new group keys from the leaves up to the root. In comparison with LKH, the bottom-up OFC and OFT methods broadcast approximately 50 percent less bits during a single member evict operation. Only OFT, however, has the option of being member contributory.

The first Internet Draft on the LKH algorithm appeared in July 1997, authored by Wallner et al. [57]. The name LKH (suggested by Sherman) was adopted by Harney and Harder [29] in their 1999 implementation of the algorithm. In July 1997, Wong et al. [58], [59] analyzed generalizations of LKH and, in 1998, Caronni et al. [18] independently published a similar idea to LKH. OFT was developed at Network Associates Laboratories (formerly, Trusted Information Systems) as part of the DARPA-funded Dynamic Cryptographic Context Management (DCCM).
Project [21], [4]. In November 1997, Sherman et al. [25] made the first presentation on OFT. The OFT algorithm is the subject of this document.

The LKH method achieves logarithmic broadcast size, storage, and computational cost. A hierarchy of keys is created, and each group member is secretly given one of the keys at the bottom of the hierarchy. Each interior key is encrypted with all of its children keys, and all of these ciphertexts are broadcast to the group. Each member can decrypt the keys along the path from their leaf to the root; the root key is used as the group key. The interior keys are associated with logical (rather than physical) security domains. Thus, after an addition or eviction, a group of n members can rekey by broadcasting about 2 lg n keys.

Importantly, in 1999, Canetti et al. [15, Section 4.2] proposed a variation of OFT, which we refer to as One-Way Function Chain (OFC). In OFC, there is always a functional relationship among the node secrets along the path in the key tree from some leaf to the root. This functional chain changes over time and will hold for the last leaf whose user was removed. OFC shares the broadcast reduction of OFT over LKH; additionally, OFC is slightly simpler than OFT.

Canetti claims (without formal written proof) that both LKH and OFC can be proven secure.

A minor additional advantage of OFC over OFT is that each member would typically store only lg n secrets rather than 2 lg n node secrets. However, in OFC, each member is required to store only 1 + lg n secrets because each member could recompute the node secrets along its path to the root; this time-space tradeoff is not possible in OFC.

In the OFT and OFC algorithms, the functional relationships are based on a special type of one-way function defined from a pseudorandom function (see Section 3.2); they are not based on arbitrary one-way functions. Thus, care must be taken not to jump to false conclusions from the convenient and commonly-used names OFT and OFC.

Related work by Canetti et al. [16] gives a generalization and improvement of LKH based on a storage-communication tradeoff for the purpose of reducing member storage. They show that, for a group of n users with user storage of b + 1 keys, rekeying can be done by broadcasting O(bn^(1/b) - b) keys with manager storage of approximately O(n). They also prove lower bounds of b + 1 member storage and n^1/b keys broadcast, which is tight for constant b. An interesting special case of their tradeoff is O(lg n) member storage, O(lg n) keys broadcast, and O(n/lg n) manager storage.

The hierarchical methods of Caronni et al. [18] and of Chang et al. [19] suffer from a serious security limitation involving bulk eviction, which makes them ill-suited for most applications. These methods assign keys to nodes of the binary key tree based on the binary representation of the identifier of the group. This key distribution reduces the key storage requirements of the group controller and enjoys the same communication requirements for single member eviction as LKH. These methods also lower communication requirements for bulk eviction in comparison with LKH. However, these methods are vulnerable to collusion between simultaneously deleted members. For example, if a member (e.g., UID = 010) and a member with complementary user identification number (e.g., UID = 101) are removed at the same time, then if they collude they will know all of the keys of the key tree and, therefore, the tree cannot be securely updated.

Additional hierarchical methods similar to LKH, OFC, or OFT continue to emerge, such as the ELK algorithm proposed by Perrig et al. [43]. A major idea of ELK is to send only part of the key; the receiver searches for the remaining part. This idea can also be applied to OFT. This idea, however, does not work for spread-spectrum applications in which the receiver needs the key to receive the transmission.

Concerning their subset-difference method, Lotspeich et al. [37] and Naor et al. [42] claim to be able to evict r members with an average broadcast length of only 1.25r keys. This method is not very practical given that r is the cumulative number of evictees. Also, vacated positions in the tree cannot be reused.

Selcuk and Sidhu [49] observe that, if the a priori member eviction probabilities are known, then these probabilities can be used to advantage by placing members with high eviction probabilities near the root of any key tree rather than associating all members with leaves. Previously, Poovendran and Baras [44] also studied similar ideas. Unfortunately, for most applications, such probabilities are likely not to be known; furthermore, estimating such probabilities online is likely not to yield much useful information.

Yang et al. [60] analyze the benefits of batch operations. The earlier communication tree based methods of Ballardie [7], [8] and Harkins and Naganand [27] are scalable but require trusted routers.

2.3 Surveys and Additional Related Work

For additional discussion of previous and related work, see our Internet Draft [6], Menezes et al. [39], our DCCM Report [4], and Briscoe [12]. In addition, Harney and Harder [28] and Canetti and Pinkas [17] provide relevant background for the requirements of group operations for multicast security.

Self-Healing key distribution by Berson et al. [10] enables a receiver to recover the known number m − 2 of missing “interior” keys in the key sequence K1, K2, ..., Km given the “exterior” keys K1 and Km. Unfortunately, the huge update broadcast requires (2n − 1)m keys for group size n.

Rodeh et al. [47] propose a tree rebalancing method and a distributed version of LKH called dLKH.

Internet Draft [6] also proposes a novel amortized group induction process due to Sherman [52] and Balenson et al. [5] that offers savings when the universe of members N is smaller than Gn, where G is the number of groups and n is the typical group size.

3 GROUP KEYING IN THE CENTRALIZED HIERARCHICAL MODEL

The centralized hierarchical methods (including OFT, LKH, OFC as introduced in Section 2.2) form a simple and scalable class of group keying methods for large groups. They use a key tree in which the leaves are associated with group members and the root is associated with the group key. This key tree is logical in the sense that it is entirely

likely exampl of ban with airpl: could adm from ' in goo tempo
3.2 Common Terminology for Hierarchical Methods

To describe the three hierarchical methods OFT, LKH, and OPC in a common framework, it is convenient to introduce the following terminology. Each node \( v \) of the key tree has a node secret \( s_v \) and a node key \( k_v \) (in LKH each node key is equal to its node secret). The group key is the root node secret (i.e., the node secret of the root node). The shared keys (see Section 3.1) are completely separate from the leaf node keys. The node keys are used to protect communications from the manager to certain members; the node secrets are used in OFT and OPC to derive the group key. It is sound security engineering to separate these two functions.

In each method, each member must know its own shared secret and all node secrets along the path from the member to the root. The methods differ primarily in how they compute the node secrets and in what information must be broadcast.

The most interesting operation is the evict operation. In each method, during an evict, all of node secrets known by the evicted member are changed; these node secrets are the node secrets for all nodes from the (leaf of the) evicted member to the root. The manager broadcasts information sufficient to enable all remaining group members to compute the newly changed node secrets, including the new group key. The OFT and OPC methods require fewer broadcast bits than does LKH by exploiting a functional relationship among certain node secrets. The logarithmic broadcast size stems from the fact that, for balanced trees, the length of the path from the evicted member to the root is logarithmic in the group size.

The OFT, LKH, and OPC hierarchical methods can be described using the following common set of cryptographic primitives.

- To wrap broadcast messages for confidentiality in transmission, each of the hierarchical methods uses a symmetric encryption function \( E \). Let \( E_k[x] \) denote the encryption of message \( x \) under key \( k \).
- To compute the node secrets and node keys, the OFT and OPC methods use two special one-way functions \( f \) and \( g \). These functions are defined as the left and right halves of a pseudorandom function; they are not simply any one-way functions. The function \( f \) is used to compute functional relationships among node secrets; the function \( g \) is used to compute each node key from its corresponding node secret. Both \( f \) and \( g \) compute “blinded” values from the node secrets in such a way that protects the confidentiality of the node secrets; for any node secret \( x \), however, we shall refer only to \( f(x) \) as the “blinded node secret.”

Adapting an approach of Canetti et al. [16] to OFT, we shall define the functions \( f \) and \( g \) in terms of a single length-doubling pseudorandom function \( H \). Specifically, given \( H \), define \( f \) and \( g \) by \( f \| g = H \), where \( \| \) denotes concatenation. For example, in practice, one might define \( H \) based on a cryptographically-secure hash function such as SHA-1 [23].

There are some subtle security advantages in using \( f \) and \( g \) as defined here, at least in terms of proving the security of the methods (see Sherman [51]).
Required conditions on the component functions $E$, $f$, and $g$ are discussed in detail in Sherman [51]. In short, sufficient conditions for the security of OFT are that $E$ is a secure encryption function and that $H$ is pseudorandom (with $f$ and $g$ defined from $H$ as explained above). It is also necessary for the security of OFT that $E$ be secure. Determining corresponding precise necessary conditions on $f$ and $g$ remains an open problem.

In a typical broadcast message from the manager, a message component consists of some node secret encrypted using a child node's node key. As discussed in Section 7.2, additional straightforward security mechanisms (e.g., authentication and message chaining) are also needed in communications from the manager to guard against active attacks.

For some applications, it is convenient to be able, for any node, to use its node key as a "subgroup key" for the subgroup of members that are descendants of the node (leaves of the subtree rooted at the node). This optional functionality is an attraction of the hierarchical model.

We note that the terminology and detailed functional relationships of OFT as described in this paper differ from those given in earlier drafts [38], [5] of this paper. For example, the $f$ function in this paper corresponds to the "blinding function" $g$ in the earlier drafts; the $g$ function in this paper is new; and the $f$ function in the earlier drafts is now simply bitwise exclusive-or. Our current formulation provides a cleaner system for which it is easier to prove security properties.

3.3 A Brief Overview of LKH, OFT, and OFC

To describe each of the hierarchical methods LKH, OFT, OFC, it is necessary to explain how the node secrets $x_v$ and node keys $k_v$ are computed and what information is broadcast by the manager. Throughout, we shall apply the primitive cryptographic functions $E$, $f$, $g$ defined in Section 3.2.

3.3.1 Computing Node Keys

In LKH, the manager chooses all node keys independently. In OFT and OFC, each node key is computed by applying the function $g$ to the node secret: For each node $v$, the node key $k_v$ is computed by $k_v = g(x_v)$. The node keys are used to protect communications from the manager. It is convenient to derive node keys from node secrets and doing so in this way enjoys certain security advantages (see Sherman [51]).

3.3.2 Computing Node Secrets

In OFT and OFC, the node secrets are used to derive the group keys in a bottom-up fashion. In the LKH key tree, each node key is equal to its node secret, and there are no functional relationships among any of the node secrets. We now explain how OFT and OFC compute node secrets.

Let $v$ be any interior node in the key tree, and let $L$ and $R$ be, respectively, the left child and right child of $v$.

In OFT, the group key is determined using all of the leaf secrets in the key tree: The group key is computed as a tree of function computations going from the leaf nodes to the root. Specifically, as shown in Fig. 1, the node secret for $v$ is computed by $x_v = f(x_L) \oplus f(x_R)$, where $\oplus$ denotes bitwise exclusive-or (XOR).

3.3.3 Eviction Broadcasts

During an eviction, the node secrets (and, hence, node keys) change for each node along the path in the key tree from the evicted member to the root. The manager broadcasts certain encrypted information to enable the group members to learn the new node keys they need to know. Let $v$ be any interior node in the key tree, and let $L$ and $R$ be, respectively, the left child and right child of $v$.

In LKH, the manager chooses the new node keys, if node $v$ is along this path, then the manager's broadcast would include the components $E_{k_L}(k_v)$ and $E_{k_R}(k_v)$.

In OFT, the manager chooses a new node secret for the evictee's sibling and recomputes the function tree to the root. Assume without loss of generality that nodes $L$ and $R$ are along the evictee's path to the root. Then, the manager's broadcast would include the component $E_{k_R}(f(x_L))$.

In OFC, the manager chooses a new secret for the evictee's parent and recomputes the function tree to the root. Assume without loss of generality that node $v$ is along the evictee's path to the root and that the chain passes through nodes $R$ and $v$ but not $L$. Then, the manager's broadcast would include the component $E_{k_v}(x_L)$.

4 One-Way Function Trees

Our new group keying method uses a special one-way function to compute a tree of keys; we call our method the One-Way Function Tree (OFT) algorithm. The keys are computed up the tree, from the leaves to the root; this approach reduces rekeying broadcasts to only about $\log n$ keys, where $n$ is the number of group members. This section describes OFT in terms of its tree structure and its initialization, addition, eviction, and batch operations.

The group manager maintains a binary key tree, each node $v$ of which has two cryptographic values: a node secret
x, and a node key \( k_v \). As explained in the next section, the node secrets are functionally related by means of a special one-way function \( f \). We shall sometimes refer to the value \( f(x_v) \) as the "blinded node secret" of node \( v \). The node secret is blinded in the sense that a computationally limited adversary can know \( f(x_v) \) and yet cannot find \( x_v \). Similarly, for each node, the node key is computed from the node secret using a special one-way function \( g \), thus, \( k_v = g(x_v) \).

The manager uses a symmetric encryption function \( E \) to communicate securely with subsets of group members. The functions \( E, f, \) and \( g \) are defined in Section 3.2; their required properties are discussed in detail in Sherman [51]. During rekeying operations, the manager communicates with the group members via broadcast messages. In addition, there is a separate external secure channel between the manager and each member used during group initialization.

Although we believe our bottom-up use of one-way function trees for group keying is novel, the idea of using one-way functions in a tree structure is not new. Merkle [40] proposed an authentication method based on such trees. Also, Fiat and Naor [22] used a one-way function in a top-down fashion in their group keying method for the purpose of reducing the storage requirements of information theoretic group key management.

4.1 Structure of an OFT

The OFT key tree is a particular type of binary tree in which each interior node has exactly two children. Every leaf of the tree is associated with a group member, and the node secret of the root is the common group key. Group members can use this group key to communicate among themselves with privacy and/or authentication or to gain access to some broadcast signal.

Before the first group key is computed, a randomly-chosen node secret is assigned to each member. A variety of choices are possible in governing who chooses these secrets. For example, the key could be chosen by the manager, member, or a combination thereof. In addition, during initialization, for each member the manager establishes a separate "shared key" known only by the manager and member.

For any interior node \( v \) in the OFT key tree, the node secret \( x_v \) of \( v \) is defined by \( x_v = f(x_L) \oplus f(x_R) \), where \( L \) and \( R \) denote the left and right children of \( v \), respectively, \( f \) is a special one-way function, and \( \oplus \) is bitwise exclusive-or.

The use of exclusive-or in the computation of node secrets does not prevent OFT from being used with structured keys. With structured keys, the unstructured OFT group key would be used as a key-encrypting key (KEK), which would encrypt the structured group key. Even for unstructured group keys, in some applications, there may be advantages in using the OFT group key as a KEK.

The security of the system depends on the fact that each member's knowledge about the current state of the key tree is limited by the following invariant:

**System Invariant.** Each member knows the node secrets on the path from its node to the root (and therefore the node keys along this path), and the blinded node secrets that are siblings to this path, and no other node secrets nor node keys.

This invariant is maintained by all operations that add or delete members.

Each group member maintains the node secret of the leaf with which she is associated, and a list of blinded node secrets for all of the siblings of the nodes along the path from her node to the root. This information enables her to compute the node secrets along her path to the root, including the root key, which she also stores. This information also enables her to compute the node keys along this path. If one of the node secrets changes and she is told the new value, then she can recompute the node secrets on her path to the root and find the new group key.

The security of the group key depends in part on the combination of blinded node secrets through the \( \oplus \) combining function and the confidentiality of broadcast communications from the manager.

The node keys provide an important functionality: Each internal node key can be used as a communications subgroup key for the subgroup of all descendant members. This functionality enables the add and evict operations to be implemented efficiently.

4.2 Initialization

Group initialization is a three-step process: First, each member establishes a "shared key" known only by the member and her group manager. Second, the manager creates the OFT key tree and assigns initial members to its leaves. Third, the initial group key is computed.

We refer to this first step of establishing shared keys as group induction, an amortized version of which is explained in [6]. This key establishment can be accomplished with any pairwise authenticated key exchange protocol, such as the Internet Key Exchange (IKE) Protocol [26].

In the second step, a variety of choices are possible for creating the OFT tree and numbering its nodes. For example, one might allow the tree to grow and shrink dynamically as members are added and evicted, as we do in the examples of this section. Alternatively, one might allocate a large complete tree (with unoccupied leaves for future members) and number the nodes using an in-order scheme, while allowing for possible future tree doubling, as we did for simplicity in our implementations. Section 7.2 discusses these issues in more detail. Having node numbers allows the option of including node numbers in manager broadcasts to facilitate a member's task of identifying the portion of a broadcast that is directed at the member.

In the third step, members compute the initial group key after receiving crucial information broadcast from the manager. To compute the group key, each member needs to learn the blinded node secret of each sibling node along the path from the member to the root. To this end, the manager broadcasts every blinded node secret in the OFT to all group members. In this broadcast, each blinded node secret is encrypted by the node key of the sibling node, so that only members in the sibling subtree can learn the blinded node secret.

All members receive the entire broadcast, which consists of a sequence of encrypted blinded node secrets. Section 7.2 discusses some of the engineering choices that must be addressed in working out the details of this broadcast. For example, the order in which the secrets are broadcast could
be determined by a preorder traversal of the OCT, as we did in our implementation. Also, at the expense of longer broadcasts, one might optionally choose to delineate some message parts of this broadcast with explicit node numbers for redundancy; alternatively, one might rely on each member to parse the broadcast stream by tree traversal order. Regardless, this broadcast enables each member to decrypt appropriate message parts to recover the blinded node secrets that she needs to know to compute the initial group key.

After receiving this initial broadcast from the manager, each member computes the initial common group key by computing the tree of function secrets as explained in Section 4.1. Each member also computes the node keys along this path to the root, because these node keys are needed to decrypt relevant message parts from the manager's broadcast.

4.3 Adding or Evicting a Member

In this section, we will describe how to add and evict members. These operations are similar: For each operation, a node secret of some leaf of the key tree changes, which affects all of the node secrets along a path from this leaf to the root. The manager securely communicates the changed information along this path to those members who need to know. The manager and all members individually compute the new group key. More specifically, if $v$ is a nonroot node along the affected path and, if $s$ is the sibling of $v$, then the manager broadcasts the blinded new node secret $f(x_v)$ of $v$ encrypted with the node key $k_s$ of $s$. Doing so enables all descendants of $s$ (and only these descendants) to learn the new $f(x_v)$ value but not $x_v$. Figs. 2 and 3 illustrate by example how the add and evict operations work and what messages are broadcast by the manager.

We shall describe the add and evict operations in terms of a dynamic key tree that grows and shrinks in size and whose leaves can be on various levels. For many applications, however, we recommend for engineering convenience that the key tree be implemented as a static complete tree with all leaves on the bottom level.

When a new member joins the group, an existing leaf node $v$ is split, the member associated with $v$ is now associated with $left(v)$, and the new member is associated with $right(v)$. Both members are given new node secrets, which will affect all node secrets along their path to the root. The old member receives a new node secret because her former sibling member knows her old blinded node secret and could use this information in collusion with another group member to find an unblinded node secret that is not on his path to the root. The new blinded node secrets that have changed are broadcast securely to the appropriate subgroups, as described in the first paragraph of this section. The number of blinded node secrets that must be broadcast to the group is equal to the distance from $v$ to the root plus two. In addition, the new member is given her set of blinded node secrets, in a unicast transmission using the external secure channel. In order to keep the height $h$ of the tree as low as possible, the leaf closest to the root is split when a new member is added. See Fig. 2.

Now, consider what happens when the member associated with any leaf $u$ is evicted from the group. Let $s$ be the sibling of $u$ and let $p$ be the parent of $u$. If $s$ is a leaf, then the member assigned to $s$ is reassigned to $p$ and given a new node secret. If $s$ is the root of a subtree, then $p$ becomes $s$, moving the subtree closer to the root, and one of the leaves of this subtree is given a new node value (so that the evictee no longer knows the blinded node secret associated with the root of the subtree). All node secrets along the path from the changed leaf node secret to the root are thereby affected. The manager securely broadcasts all new blinded node secrets that have changed to the appropriate subgroups, as described in the first paragraph of this section. The number of node secrets that must be broadcast is equal to the distance from $u$ (or a leaf of the subtree of $s$) to the root. See Fig. 3.

Fig. 2. Inserting a new member into an OCT key tree. For each node $v$, $x_v$ is the node secret of $v$, $k_v$ is the node key of $v$, and $f(x_v)$ is the blinded node secret of $v$. Via a unicast, the newly added Member $M$ is informed of the blinded node secrets of her ancestral siblings $v_1, v_2, v_3$. She also computes the new unblinded keys of her ancestors $v_1, v_2, v_3$. Via a multicast, the other members who need to know are informed of the blinded node secrets $f(x_v)$ at nodes along the new member's path to the root. To accomplish this task, the manager broadcasts the following three encrypted messages: $E_{k_1}[f(x_1)], E_{k_2}[f(x_2)],$ and $E_{k_3}[f(x_3)]$, where $E$ is an encryption function and $f$ is a special one-way function.

Fig. 3. Adding a new node to an OCT key tree.
4.4 Multiple Addition and Eviction

The broadcast size and computational effort of multiple additions and evictions can be substantially reduced by using a batch operation that evicts and/or adds multiple members rather than repeatedly applying individual add or evict operations. This reduction, which is analyzed in Section 6.3, stems from the fact that a set of individual operations may repeatedly change node secrets along common segments of the key tree.

Consider what node secrets must change after a subset of members is added or evicted. The nodes whose node secrets change are on the tree of ancestors of the affected leaves; we call this tree the Combined Ancestor Tree (CAT) of the subset of members. After a batch operation, the manager must broadcast the blinded secrets of all nodes on the CAT; the size of this broadcast is the size of the CAT. An economy of scale happens when the size of the CAT is less than the number of affected members times the height of the key tree.

Propagating the required changes through the tree and computing the broadcast information can be done in a postorder traversal of the CAT. The CAT can be computed while it is traversed.

Batch operations are natural in many applications, and they arise in the enhanced versions of OFT and LKH discussed in the next section.

5 ENHANCED OFT+ AND LKH+ ALGORITHMS

As suggested in an observation attributed to Radia Perlman, the add-member operation in LKH can be significantly improved to run in constant time. The method is to compute the new group communications key by applying a suitable one-way function to the current group communications key, to unicast the group key to the new member, and to broadcast that this update has been done. We shall refer to this refinement of LKH, which maintains perfect backward security, as LKH+.

A similar refinement is also possible for OFT (and OFC), which we shall call OFT+ (and OFC+). For each of these enhanced algorithms, the add-member operation advances the group communications key and marks (i.e., places on a list) the specified member to be added at the next full rekey (e.g., typically at the next evict-member operation). Thus, the work of adding new members is deferred until the next rekey and not eliminated. Two savings, however, are possible. First, there is an economy of scale when adding multiple members, especially when the new members will be placed compactly in the same section of the key tree. Second, a newly added member might be evicted before the next rekey.

The cost of the deferred add-member operations is given by the size of the CAT (Combined Ancestor Tree) of the members (see Section 4.4). In practice, the cost of adding $R$ new members at once (assuming they will be placed compactly in the same section of the tree) is approximately $2R + \lg(n/R)$, where $n$ is the number of nodes in the key tree.

Unfortunately, this refinement does not apply to the delete-member operation. Fortunately, in many applications, add-member is performed more frequently than evict-member.

Independently of Perlman, Viswanathan [56, pp. 25-26] suggested a similar constant-time member-parallel add-member idea. Briscoe's [12] MARKS method also applies similar ideas. The related "improvement" of LKH+ proposed by Rafaeli et al. [45], however, has a possible security weakness resulting from its selection of encryption keys that are related to the plaintext being encrypted.
TABLE 1
Comparison of the Maximum Computational and Transmission Requirements for the SDKC, LKH, and OFT Group Keying Methods

### Initializing Group

<table>
<thead>
<tr>
<th></th>
<th>SKDC</th>
<th>LKH</th>
<th>LKH+</th>
<th>OPT</th>
<th>OFT+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broad. size (bits)</td>
<td>nK</td>
<td>2nK + h</td>
<td>2nK + h</td>
<td>2nK + h</td>
<td>2nK + h</td>
</tr>
<tr>
<td>Manager comp.</td>
<td>n(Ce + C_f)</td>
<td>2n(Ce + Ce)</td>
<td>2n(Ce + Ce)</td>
<td>2n(Ce + Ce) + nC_f</td>
<td>2n(Ce + Ce) + nC_f</td>
</tr>
<tr>
<td>Max. mem. comp.</td>
<td>Ce</td>
<td>hCe</td>
<td>hCe</td>
<td>hCe</td>
<td>hCe</td>
</tr>
</tbody>
</table>

### Adding a member

<table>
<thead>
<tr>
<th></th>
<th>SKDC</th>
<th>LKH</th>
<th>LKH+</th>
<th>OPT</th>
<th>OFT+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broad. size (bits)</td>
<td>nK + ln n</td>
<td>2hK + h</td>
<td>h</td>
<td>hK + h</td>
<td>h</td>
</tr>
<tr>
<td>Unicast size</td>
<td>-</td>
<td>k</td>
<td>k</td>
<td>k</td>
<td>k</td>
</tr>
<tr>
<td>Manager comp.</td>
<td>nC_e + C_f</td>
<td>h(2Ce + C_f)</td>
<td>Ce + C_f</td>
<td>h(Ce + 2CF) + Ce</td>
<td>C_f + Ce</td>
</tr>
<tr>
<td>Max. mem. comp.</td>
<td>Ce</td>
<td>hCe</td>
<td>Ce</td>
<td>hCe</td>
<td>Ce</td>
</tr>
</tbody>
</table>

### Adding l members

<table>
<thead>
<tr>
<th></th>
<th>SKDC</th>
<th>LKH</th>
<th>LKH+</th>
<th>OPT</th>
<th>OFT+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broad. size (bits)</td>
<td>(n+l)K</td>
<td>2(lK + lh)</td>
<td>lh</td>
<td>sK + lh</td>
<td>lh</td>
</tr>
<tr>
<td>Unicast size</td>
<td>-</td>
<td>lK</td>
<td>lK</td>
<td>lK</td>
<td>lK</td>
</tr>
<tr>
<td>Manager comp.</td>
<td>(n+l)(Ce + lC_f)</td>
<td>h(2Ce + C_f) + C_f</td>
<td>h(2Ce + C_f) + C_f</td>
<td>C_f + Ce</td>
<td>C_f + Ce</td>
</tr>
<tr>
<td>Max. mem. comp.</td>
<td>C_e</td>
<td>hCe</td>
<td>C_f</td>
<td>hCe</td>
<td>C_f</td>
</tr>
</tbody>
</table>

### Evicting a member

<table>
<thead>
<tr>
<th></th>
<th>SKDC</th>
<th>LKH</th>
<th>LKH+</th>
<th>OPT</th>
<th>OFT+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broad. size (bits)</td>
<td>nK</td>
<td>2hK + h</td>
<td>2hK + h</td>
<td>hK + h</td>
<td>hK + h</td>
</tr>
<tr>
<td>Manager comp.</td>
<td>nC_e</td>
<td>h(2Ce + C_f)</td>
<td>h(2Ce + C_f)</td>
<td>h(Ce + 2C_f) + Ce</td>
<td>h(Ce + 2C_f) + Ce</td>
</tr>
<tr>
<td>Max. mem. comp.</td>
<td>Ce</td>
<td>hCe</td>
<td>hCe</td>
<td>hCe</td>
<td>hCe</td>
</tr>
</tbody>
</table>

### Evicting l members

<table>
<thead>
<tr>
<th></th>
<th>SKDC</th>
<th>LKH</th>
<th>LKH+</th>
<th>OPT</th>
<th>OFT+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broad. size (bits)</td>
<td>(n-l)K</td>
<td>(2s-l)K + lh</td>
<td>(2s-l)K + lh</td>
<td>slh</td>
<td>slh</td>
</tr>
<tr>
<td>Manager comp.</td>
<td>(n-l)(Ce + C_f)</td>
<td>h(2Ce + C_f)</td>
<td>h(2Ce + C_f)</td>
<td>s(Ce + 2C_f) + Ce</td>
<td>s(Ce + 2C_f) + Ce</td>
</tr>
<tr>
<td>Max. mem. comp.</td>
<td>Ce</td>
<td>hCe</td>
<td>hCe</td>
<td>hCe</td>
<td>hCe</td>
</tr>
</tbody>
</table>

For LKH+ and OFT+, if R add-member actions have been deferred, then the evict-member operation will additionally broadcast [2R ln (n/R)]K bits. See Section 6.3.

These numbers are approximate. Here, n is the number of members in the group, h is the height of the key tree, s is the size of the CAT when leaves change, and K is the size of a key in bits. Also, Ce, C_f, and C_f are respectively the computational cost of one evaluation of the encryption function E, generating one key from a cryptographically-secure random source, and of one evaluation of the one-way function f.

6 Computation, Transmission, and Storage Requirements

In this section, we distill the most important differences among the three methods, as revealed by Tables 1 and 2; we also analyze the costs of the batch operations.

Tables 1 and 2 give the complexities of the group initialization, key establishment, and rekeying tasks. These tasks are independent of the final step of carrying out secure group communications with an established key. Importantly, while both LKH and OFT each require only a logarithmic number of bits (in the group size) to be broadcast to rekey after an add or evict operation, OFT roughly halves the broadcast length over LKH through its bottom-up approach. In typical multicast environments, where no unicast is possible, the add-member unicast messages of LKH+, OFT, and OFT+ would be sent as
TABLE 2

Storage Requirements of the Group Manager and Group Members, for the SKDC, LKH, and OFT Group Keying Methods

<table>
<thead>
<tr>
<th></th>
<th>SKDC</th>
<th>LKH</th>
<th>LKH+</th>
<th>OFT</th>
<th>OFT+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager storage</td>
<td>$nK$</td>
<td>$2nK$</td>
<td>$2nK$</td>
<td>$2nK$</td>
<td>$2nK$</td>
</tr>
<tr>
<td>Member storage</td>
<td>$2K$</td>
<td>$hK$</td>
<td>$2hK$</td>
<td>$hK$</td>
<td>$2hK$</td>
</tr>
</tbody>
</table>

Here, $n$ is the number of members in the group, $h$ is the height of the key tree, and $K$ is the key size in bits.

multicasts. In such environments, the broadcast savings of OFT over LKH are for the evict-member operation only.

6.1 Terminology

Our analysis assumes that the trees used by LKH and OFT are binary. Broadcast sizes increase with the branching factor of the tree, so binary trees minimize the communication cost, and binary trees are a practical choice. For readability, all of the values neglect constant additive factors. We express these values in terms of constants which represent basic requirements of cryptographic primitives $f$ and $E$ and the size $K$ of one key in bits. Thus, $C_E$, $C_f$, and $C_b$ are, respectively, the computational cost of one evaluation of the encryption function $E$, generating one key from a cryptographically secure random source, and of one evaluation of the one-way function $f$. The number of members in the group is $n$.

In the LKH and OFT methods, the manager must notify the members about the changes in the topology of the tree; this requirement is reflected in our analysis. The number of bits required to specify a node in a binary tree is equal to the height $h$ of the tree.

The performance of LKH and OFT depends on the height of the key tree. If dynamic trees are used, to minimize $h$, new members can be added as close to the root as possible. If many evictions occur, the resulting tree may be "unbalanced," that is, its height may be much greater than $\log n$. A rebalancing operation can be performed to restructure the tree so that its height is no greater than $\log n + 2$.

6.2 Quantitative Comparison of SKDC, LKH, LKH+, OFT, and OFT+

The broadcast size to rekey after an addition or eviction is $nK$ bits ($n$ keys) for SKDC, $2nk + hK + h$ bits for LKH, and $2nk + h$ for OFT. OFT achieves a smaller broadcast than LKH through its bottom-up approach. The extra factor of 2 in LKH appears because in LKH two separate encryptions are needed per internal node—one for each of the two children. In both LKH and OFT, the "$+h$" term is to specify which node was added or evicted.

The computational, storage, and broadcast costs of OFC are comparable to those of OFT. However, should one ever need to reestablish the chain in OFC, that nonstandard operation could be done in $h$ steps, which is much faster than the $n$ steps required to reestablish the one-way function tree in OFT.

For OFT, the security and timing analysis flows from the invariant properties preserved by each OFT operation. In particular, to compute the group key, each member needs to know: her own key and her blinded ancestral sibling node secrets (there are $h$ of the latter). During eviction, the evicted slot is pruned and a leaf node along the resulting path is changed. These changes propagate up the tree. A broadcast of $h$ keys is required to update all the other members who had depended on the blinded node secrets which have changed. Nothing else needs to be broadcast or changed (the blinded node secrets the evicted party knew are useless).

During an OFT add, the situation is similar but slightly different. To add a new member, we expand a leaf node, announce the addition, give the new member a key, and then we do the following two steps: 1) As with evict, since the new member's key propagates up the tree, we need to broadcast to everyone who depends on that information the new blinded ancestor node secrets. 2) Unlike evict, we also have to unicast to the new member the blinded ancestral sibling keys she needs to know. The factor of two savings in broadcast is for both add and evict, but add and not evict also requires an additional $h$ keys in unicast.

The broadcast size to initialize the SKDC is $n$. The LKH and OFT require about twice that size of broadcast since every key in a binary tree with $n$ leaves (and, therefore, $n - 1$ internal nodes) must be communicated.

The storage requirements of both LKH and OFT members are about $h$ keys, while that of SKDC members is exactly two keys. The storage requirement of LKH and OFT managers is about $2n$, while that of SKDC managers is exactly $n + 1$.

Another advantage of OFT over LKH is that OFT requires significantly fewer random bits. In particular, to add one member, in OFT, the manager must generate only one new random key. By contrast, in LKH, the manager must generate $h$ new keys. If key generation is performed in software, this difference could yield OFT a significant time advantage over LKH since in some practical applications generating random bits is relatively slow.

The LKH and OFT distribute the computational cost of rekeying among the whole group, so that the group manager's burden is comparable to that of a group member. This characteristic makes these algorithms especially appropriate for an online system.

6.3 Analysis of Multiple Addition and Eviction

In the bulk eviction and bulk addition operations, the broadcast size and the computational effort depend on the size of the CAT (see Section 4.4). We compute upper and lower bounds on the size (number of nodes) $s_l$ of a CAT with $L$ leaf nodes, yielding bounds on the broadcast size and computational effort to add or evict a set of $L$ members.

Intuitively, the minimum CAT size occurs when the CAT is a densely embedded subtree of the key tree, as would happen when the members to be processed are adjacent leaves of the key tree. The maximum size occurs when the CAT is sparsely embedded in the key tree, as occurs when the members to be processed are evenly distributed as leaves.

More specifically, the minimum value of $s_l$ occurs when the CAT includes a "densely packed" subtree of size $2L - 1$ at the lowest level of the key tree, attached to the root of the key tree by a single path of length $\log_2(n/L)$. Conversely, the maximum value of $s_l$ occurs when the CAT includes a
7 Design Choices, Implementation Issues, and Experience

In this section, we identify important design choices required to implement the OPT algorithm and discuss some of the lessons we learned from our prototype implementations.

7.1 Design Choices

Several important engineering decisions must be made in the implementation of the OPT algorithm: the choice of the length-doubling pseudorandom function $H$, the format of broadcasts by the manager, the representation and numbering of the key tree, and time-space tradeoffs by each member involving how many ancestor node secrets to store.

The keyless pseudorandom function $H$ can be based on a cryptographic hash function such as MD5 [46] or SHA-1 [23]. It is possible that the node keys do not need to be as large as the output size of the underlying hash function. For example, MD5 has a 16-byte output, while DES keys are only seven bytes long. The functions $f$ and $g$ can be constructed from MD5 by discarding some of the output, as is done by S/KEY. Note that $H$ should not be based on a keyed Message Authentication Code (MAC) because either secret keys would have to be securely distributed, or with known keys MACs would not have the pseudorandom properties desired for $H$.

Since the group key in OPT is computed as a composition of one-way functions (rather than one-way permutations), there is some coalescing of effective entropy (see Sherman [51]). Therefore, it is important to size the parameters appropriately to achieve the desired effective entropy.

The representation of the tree and the format of the messages from the group manager are important engineering decisions. For example, the tree could be represented as a record and pointer structure, or as a linear array. The tree could be static or dynamic (growing or shrinking as members are added or deleted). Assignments of members to leaves could be fixed or changeable. An important decision regarding message formats is whether and how frequently to include node number information to identify which parts of a message correspond to which subtrees. Also, one must decide in what order the messages are broadcast and how they are formatted and blocked.

Many engineering choices beyond the group keying algorithm affect system performance. For example, in selecting the block size for broadcast messages, one must balance the competing concerns of minimizing the number of garbled blocks due to transmission errors and minimizing the number of bits of overhead sent, for example, as block headers. Also, prudent use of extra broadcast channels (if available) can significantly reduce rekeying times or enhance reception.

In a complete system application, one might like to support a variety of additional operations and optimizations. For example, one might like to allow members to leave the group temporarily without losing their security privileges. Also, one might like to allow a newly added member to be able to read a limited amount of previous group communications. We suggest that this latter functionality be effected through a separate key repository mechanism that is independent of the key establishment algorithm.

7.2 Implementation Issues and Experience

The initial NAI prototype implementation of the OPT+ algorithm was carried out in 1999 for the DCCM OPT Toolkit [21], [4]. The purpose of this Java implementation is to demonstrate proof of concept and to gain insight into implementation issues.

This prototype uses the following parameter choices: The node key size is 128 bits, and $f$ and $g$ are SHA-1 with 160 bits of output each.

For simplicity, and to avoid ever having to change a member's node number in the key tree, the prototype uses a static tree with in-order node numbering. Thus, the key tree is constructed with all leaves always on the lowest level and, initially, the key tree is allocated for a certain maximum size. To grow beyond this size, the tree could be easily doubled in size by making the current tree the left subtree of a new tree and broadcasting the new root node number without renumbering any existing nodes.

The NAI prototype raised four important implementation issues: adapting messaging for unreliable communications, node numbering, caching, and node storage explosion.
proposal for its simplicity, broadcast size (similar to OFT), and provable security. In addition to being slightly simpler than OFT, OFC takes only logarithmic (rather than linear) time to reestablish the functional relationships in the key tree, should that need to be done. There may be some situations where it might be convenient to suspend the functional relationships temporarily.

The exact savings in broadcast size of OFT over LKH depends on the parameter choices in the OFT and LKH implementations, including the key lengths and the output sizes of the one-way function and encryption function. Although our current implementation does not do so, if the one-way function expanded its input more than does the encryption function, then the savings in broadcast size would be less than a factor of two. For example, using 128-bit group keys and the 160-bit SHA-1 one-way function, our initial prototype OFT achieved a savings in broadcast size over LKH of a factor of $2 \times 128/160 = 1.6$, which is slightly less than two.

Although a reduction in broadcast size of one half may seem relatively small, we conjecture that the performance of the LKH algorithm may be approaching theoretical limits and therefore only relatively small improvements may be possible. Canetti et al. [16] give some evidence supporting this conjecture by proving upper and lower bounds on the communication cost with a gap of at most $O(b) \leq O(\log n)$, where $b$ is a parameter (see Section 2).

Refinements to the OFT, OFC, and LKH methods which we call OFT+, OFC+, and LKH+ (see Section 5), offer significant performance improvements. In these refinements, the cost of the add-member operation is one one-way function application, a unicast of one key, and one short broadcast. There is also a deferred cost for the next rekey operation. These improvements make OFT+, OFC+, and LKH+ attractive choices for many applications.

It is possible to implement the OFT algorithm with $k$-ary trees, as studied in LKH by Canetti et al. [16]. The optimal choice of $k$ depends on what cost metric is to be minimized. The choice $k = 2$ minimizes broadcast size for single member eviction. By contrast, to minimize total manager computation, Wong et al. [59] show experimentally that $k = 4$ is best in LKH; we suspect that a similar result may also hold for OFT.

Section 6.3 analyzes savings that are possible by separately batching multiple add or multiple evict operations. Similar and possibly even greater savings may be possible by batching adds and evicts together (for one study, see Yang et al. [60]).

It is important to realize that there are significant fundamental limitations to achieving security in large groups—one might even say that a secure large group is an oxymoron. In most large groups, it is very likely that at least one member is unreliable, untrustworthy, malicious, or careless. Each member knows the common communications key and the plaintext, which is the main commodity being protected. Using multiple communications keys for different subgroups would not enhance security since each member would still have the plaintext. Any member could disclose the plaintext. Consequently, in large groups, it becomes especially important to detect traitors (e.g., through fingerprints and watermarks [36], [53], [20]) and to limit the loss caused by detected and nondetected disclosures (e.g., by rapid evictions, rekeying, and audits). Special-purpose, physically-secure hardware can play a role in these objectives, by restricting access to communication keys, complicating effective use of compromised keys, and providing unique fingerprints.

Our security comments on OFT raise some interesting questions about the security of function iterates and that of bottom-up one-way function trees. We leave as an open problem a rigorous statement and proof of the necessary and sufficient conditions on the functions $f$, $g$, $H$, and $E$ to ensure security of the method. Formally proving the security of key-establishment algorithms is a difficult problem. For example, Bellare and Rogaway's [9] treatment of the two-party case illustrates some of many technical obstacles that must be overcome. The multiparty case is even more difficult, and we are unaware of anyone extending the work of Bellare and Rogaway to the $n$-party case. Nevertheless, as explained in Sherman [51], we believe that the security of OFT can be formally proven. Similarly, we believe that the OFT, OFC, and LKH methods have essentially equivalent security, though the exact requirements on the underlying primitive functions may be slightly different.

A possible security concern for OFT and OFC is the loss of effective entropy in the group key caused by repeated function applications. Applying a theorem of Flajolet and Odlyzko [24, p. 336], Sherman [51] estimates, that, for 10 million users, the loss of effective entropy for OFT and OFC is approximately four bits.

An important open problem is how to perform bulk operations efficiently on multiple independent logical subgroups that do not directly reflect the structure of the key tree. For example, it may be useful to add or evict in bulk based on overlapping criteria such as geographic location, nationality, or member type.

Also, it would be interesting to identify an important application in a centralized model that could benefit from OFT's member-contributory feature.

Throughout, we have viewed the key tree as a homogeneous OFT tree whose leaves are associated with individuals. More general approaches are possible, as discussed by Sherman and McGrew [50, Section 8]. Such generalizations, which include hybrid trees, multiple trees, and logical subgroups, may be useful in applications for which the members have differing needs and characteristics. For example, some members may have different hardware, and some members may be on submarines, airplanes, or foot.

Our OFT algorithm offers a practical approach with low broadcast size to manage the demanding key establishment requirements of secure applications for large dynamic groups.

ACKNOWLEDGMENTS

The authors would like to acknowledge contributions by several colleagues at Network Associates Laboratories (NA Labs)—formerly, Trusted Information Systems (TIS). They are very grateful to Michael V. Harding for significant

REFERENCES

[1]
To implement the OFT algorithm in a multicast environment requires some messaging adaptations. The finite maximum transmission unit in this connectionless environment requires that messages be sent in packets. UDP packets can arrive out of order or fail to arrive. All messages need to be protected for confidentiality, integrity, and authentication. The prototype protects its internal messages using Blowfish encryption and IPsec-based integrity and authentication checks [33].

As an example of a messaging implementation detail, note that to encrypt a 160-bit SHA-1 output with a 128-bit block cipher requires some type of chunking. The prototype applies the Blowfish algorithm in ECB mode with no padding to 128-bit data blocks. These blocks are protected using the standard IPsec-based authentication and integrity mechanisms for all OFT messages; alternatively, one might use some type of block chaining (e.g., CBC mode).

During the course of OFT operations, it is necessary to refer to members by their names and to refer to key tree nodes by their topological positions. Also, it is necessary to be able to map between member names and their associated nodes. To this end, it is convenient to maintain node numbers. Moreover, since there is no efficient reliable way to inform members of any changes in their node numbers, it is especially convenient if the node numbers never change. For these reasons, in the prototype, member names are equated with node numbers, and node numbers never change. The prototype numbers all key tree nodes with an in-order numbering scheme. This choice enables leaves to be identified solely by their node numbers, and the tree can be doubled in size without changing any node numbers.

Broadcast messages are sent by a preorder traversal of the key tree. To minimize broadcast size, one could omit node numbers in most manager broadcasts and rely on the implicit ordering of the messages from the preorder traversal. Alternatively, for reliable communications and to enable members to listen to only a portion of broadcast messages, one could redundantly include a node number in each message part. For many applications, we recommend a tradeoff between the two extremes, as we adopted in our prototype using occasional node number checkpoints. Note that the position of a member in the key tree significantly affects when that member receives an update, and such position information might be useful to an adversary who may attempt to jam the broadcast signal.

Because OFT is intended for very large groups, the amount of storage allocated per node in the key tree significantly affects the total amount of memory used by the manager. We refer to this issue as the manager's node storage explosion. Care must be taken not to squander memory on elaborate node structures. For example, on space considerations alone, the tree-balancing strategies of Moyer et al. [41] are unlikely to be wise engineering choices for today's technology.

We found that prudent use of node caching to reduce slow disk accesses greatly enhances the performance of the algorithm (a simple caching method sped up our implementation by a factor of at least 10 to 50).

In addition, the following two lessons were learned. First, it is important but nontrivial for a member to verify when she has the correct group key. A member might compute the wrong group key for two reasons: The member might never receive some key-update messages. Also, in a sequence of key-update messages, the member might not have any way of knowing that additional key-update messages are yet to come. These situations are complicated by the unreliable nature of the multicast environment. We view these key synchronization issues as orthogonal to OFT; that is, these issues exist in many communications systems for a variety of reasons and should be solved as part of the standard key management service. For example, one solution is to attach confidential key checksums to keying data, which is what the prototype does. Specifically, key-update broadcasts include a checksum that is $E_k(k)$, where $k$ is the group key and $E$ is the Blowfish encryption. Another solution, which the prototype also uses, is to allow members to send out-of-band resynchronization request messages to the group manager.

Second, it is important that the components be appropriately matched and coordinated. For example, inefficiencies can arise if the keysize is not a multiple of the block size of the encryption function used for confidentiality.

Our current prototype has handled groups of up to 10 million members on a 400 MHz Pentium II with 256 MB of RAM and 928 MB on disk, with minor adjustments to the swap space and using 192-bit keys. The maximum possible group is limited by two factors: the size of the initial message and the manager's node storage explosion. Our implementation demonstrates that the OFT method is realizable for large groups using standard commercial computer equipment.

8 Conclusions

We have presented and analyzed a new practical centralized hierarchical algorithm for establishing shared cryptographic keys for large, dynamically changing groups. Our algorithm is based on a novel application of One-Way Function Trees (OFTs), taking a bottom-up approach with the option of member contributions to the entropy of the common communications key.

Unlike previously proposed solutions based on information theory, hybrid approaches, or a single key distribution center, our OFT algorithm has communication, computation, and storage requirements that scale logarithmically with group size, for the add or evict operation. By contrast, each of the aforementioned methods scales linearly or worse (see Section 2). Furthermore, the OFT algorithm avoids the expensive public-key operations of decentralized group Diffie-Hellman approaches and their variants.

In comparison with the first proposed hierarchical method—the Logical Key Hierarchy (LKH) [57]—our OFT algorithm reduces by approximately half the number of bits broadcast by the manager per add or evict operation. OFT is the first of the hierarchical methods to achieve this reduction in broadcast length. The user time and space requirements of OFT and LKH are roughly comparable. For many applications, including multicasts and satellite broadcasts, minimizing broadcast size is especially important.

The One-Way Function Chain (OFC) variation of OFT recently due to Canetti et al. [15] is a very attractive
contributions to the original algorithm and for discussing implementation strategies with them. Dennis K. Branstad, original Principal Investigator of the Dynamic Cryptographic Context Management (DCCM) Project, suggested the problem and provided constructive recommendations and technical guidance. They thank Pete Dinsmore and David Balenson, subsequent Principal Investigators of the DCCM Project, for management and support. They also thank William Byrd, Pete Dinsmore, Michael Ferguson, Mike Heyman, Jim Horning, Caroline Sace, Jay Turner, and the referees for helpful comments; Sharon Osuna for editorial suggestions; and Matt Mundy for incorporating the figures electronically. The prototype OBT implementation was carried out by Pete Dinsmore, Michael Ferguson, Mike Heyman, Peter Kraus, Matt Mundy, and Caroline Sace at NA Labs in Glenwood, MD. Support for this research was provided in part by the Defense Advanced Research Projects Agency under contract F30602-97-C-0277. Preliminary versions of this paper appear as NA Labs and TIS Labs technical reports [50], [38] and as an unrefereed NA Labs journal article [5]; a companion Internet Draft [6] proposes the OBT algorithm to the Internet community. This work was initially carried out while A.T. Sherman was on leave from the Department of Computer Science and Electrical Engineering at the University of Maryland, Baltimore County (UMBC).

REFERENCES


