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

David A. McGrew and Alan T. Sherman†
Cryptographic Technologies Group
TIS Labs at Network Associates, Inc.
Glenwood, Maryland 21738
email: {mcgrew, asherman}@tis.com

May 20, 1998

Abstract

We present and analyze a new algorithm 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 previously published methods, our algorithm achieves a new minimum in the number of bits that need to be broadcast to members in order to re-key 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. Our algorithm provides complete forward and backwards security: newly admitted group members cannot read previous messages, and evicted members cannot read future messages, even with collusion by arbitrarily many evicted members.

This algorithm offers a new scalable method for establishing group session keys for secure large-group applications such as electronic conferences, multicast sessions, and military command and control.

Keywords. Conference keying, cryptography, cryptographic protocols, key agreement, key establishments, one-way functions, one-way function trees, secure conferences, secure group applications.

*Support for this research was provided by the Defense Advanced Research Projects Agency under contract F30602-97-C-0277.

†On leave from the Department of Computer Science and Electrical Engineering at the University of Maryland, Baltimore County (UMBC).

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. Communication costs must also be considered.

Real-time applications, such as secure audio and visual broadcasts, pay TV, secure conferencing, and military command and control, need very fast re-keying so that changes in group membership are not disruptive. To deal with large group sizes (*e.g.* 100,000 members), we seek solutions whose re-keying 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 radio broadcast and Internet multicast.

We present and analyze a new practical algorithm 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. In comparison with previously published methods, our algorithm achieves a new low in the required broadcast size.

The rest of this paper is organized in seven sections. Section 2 briefly reviews previous approaches to group key establishment. Section 3 describes our proposed method, and Sections 4 and 5 respectively analyze its security and performance. Section 6 discusses extensions of the basic method to manage subgroups. Section 7 gives some implementation notes, and Section 8 summarizes our conclusions.

2 Previous Work

A variety of 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 the logical key hierarchy (LKH).

The simplest solution is 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 [6]. 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.

Information theoretic approaches must satisfy the memory lower bounds of Blundo *et al.* [1] and use storage exponential in group size. All of the previously published hybrid approaches that achieve perfect forward secrecy also scale at least linearly in group size [3].

GDH approaches, including those by Burmester and Desmedt [2] and Steiner *et al.* [12, 13], require a linear number of public-key operations, which are slow in software relative to encryption or one-way function operations.

The LKH method [14, 15] 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. This system allows the group to re-key after an addition or eviction with a broadcast of about $2 \lg n$ keys.

3 One-way Function Trees

The new group keying method uses one-way functions to compute a tree of keys, which we call the One-way Function Tree (OFT) algorithm. The keys are computed up the tree, from the leaves to the root; this approach reduces re-keying broadcasts to only about $\lg n$ keys,

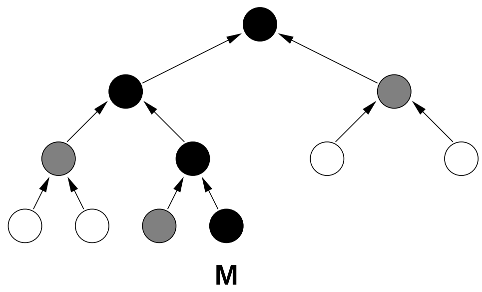

Figure 1: An example of a key tree. The member at the leaf labeled M knows only the keys of the black nodes (including the root key, which is used as the group key) and the blinded keys of the grey nodes.

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 [9] proposed an authentication method based on such trees. Fiat and Naor [3] 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.

The group manager maintains a binary tree, each node x of which is associated with two cryptographic keys, a *node key* k_x and a *blinded node key* $k'_x = g(k_x)$. The blinded node key is computed from the node key using a one-way function g; it is blinded in the sense that a computationally limited adversary can know k'_x and yet cannot find k_x.

The manager uses a symmetric encryption function E to communicate securely with subsets of group members.

3.1 Structure of an OFT

Interior nodes of the tree have exactly two leaves. Every leaf of the tree is associated with a group member. The manager assigns a randomly chosen key to each member, securely communicates this key to the member (using an external secure channel), and sets the node key of the member's leaf to the member's key. The interior node keys are defined by the rule

$$k_x = f(g(k_{\mathrm{left}(x)}), g(k_{\mathrm{right}(x)})), \tag{1}$$

where left(x) and right(x) denote the left and right children of the node x. The function g is one-way, and the function f is a "mixing" function. The properties of f and g will be discussed in detail in Section 4.1. The node key associated with the root of the tree is the *group key*, which the group can use to communicate with privacy and/or authentication.

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 unblinded node keys on the path from its node to the root, and the blinded node keys that are siblings to its path to the root, and no other blinded or unblinded keys.*

This fact is maintained by all operations that add members to or delete members from the group. An example OFT is shown in Figure 1, which illustrates what a group member knows.

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

The security of the group key results primarily from the mixing function f. The addition of the one-way function g to blind internal node keys gains an important functionality: each internal node key can be used as a communications subgroup key for the subgroup of all descendent members. This functionality is important to the efficiency of the add and evict operations.

3.2 Adding or Evicting a Member

The operations of adding and evicting members both rely on the communication of new blinded key values, from the manager to all of the members, after the node key associated with a leaf has changed. Each blinded node key must only be communicated to the appropriate subset of members to maintain security. If the blinded key k'_x changes, then its new value must be communicated to all of the members who store it. These members are all associated with the descendants of the sibling s of x, and they all know the unblinded node key k_s. The manager encrypts k'_x with k_s before broadcasting it to the group, providing the new value of the blinded key to the appropriate set of members, while keeping it from other members.

When a new member joins the group, an existing leaf node x is split, the member associated with x is now associated with left(x), and the new member is associated with right(x). Both members are given new keys. The old member gets a new key because her former sibling member knows her old blinded node key and could use this information in collusion with another group member to find an unblinded key that is not on his path to the root. The new values of the blinded node keys that have changed are broadcast securely to the appropriate subgroups, as described above. The number of blinded keys that must be broadcast to the group is equal to the distance from x to the root plus two. In addition, the new member is given her set of blinded node keys, 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.

When the member associated with the node y is evicted from the group, the member assigned to the sibling of y is reassigned to the parent p of y and given a new leaf key value. If the sibling s of y 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 key (so that the evictee no longer knows the blinded key associated with the root of the subtree). The new values of the blinded node keys that have changed are broadcast securely to the appropriate subgroups, as described above. The number of keys that must be broadcast is equal to the distance from y to the root.

3.3 Multiple Addition and Eviction

The broadcast size and computational effort of multiple additions and evictions can be substantially reduced by using an operation that evicts and adds multiple members. This operation relies on the communication of new blinded node keys to appropriate members after many leaf nodes change. The nodes whose blinded keys change are on the tree of ancestors of the changed leaves; we call this tree the *Common Ancestor Tree (CAT)*. The size of this tree is the number of blinded node keys that must be recomputed and broadcast to the group.

The algorithm for propagating the changes through the tree and computing the broadcast information is a post-order traversal of the CAT, from the lowest level to the highest. The CAT can easily be computed during the traversal.

4 Security Analysis

Evicted members have some information about the key tree but not enough to compute directly any unblinded node key. Importantly, arbitrary coalitions of evicted members cannot directly compute any unblinded node keys. The unpredictability of OFT group keys also depends on the random selection by the manager of the leaf keys.

Definition 1. *(Security) The OFT method is secure if it is computationally infeasible for an adversary to compute any unblinded node key with non-negligible probability.*

Note that if an adversary can compute an unblinded node key, she can use it to decrypt other blinded node keys. We therefore require *all* unblinded node keys to be uncomputable, to protect the group key.

After a member is evicted, the keys along the path from her node to the root change. Because of the system invariant described in section 3.1, she only knows the blinded keys of the siblings to the path from her parent to the root. These blinded keys are insufficient to compute directly any unblinded keys.

The most recently evicted member cannot gain any advantage by colluding with any previous evictee: Let k' be any valid blinded key known by the previous evictee after his eviction. If k' is a sibling of a valid blinded key know by the most recent evictee after his eviction, then k' changed during the last eviction.

If multiple members are evicted simultaneously, all of the keys of the nodes on the CAT change. The sum total of the evictees' knowledge about the key tree is all of the blinded keys of nodes that are siblings of nodes in the CAT but are not in the CAT; we call these nodes CAT-siblings. The evicted members do not know any unblinded keys that are on sibling nodes, so they cannot compute any unblinded node keys and cannot compute the group key directly.

It is a security advantage of OFT that the manager selects all the leaf keys. Doing so prevents the enemy from maliciously trying to create collisions in the one-way function.

4.1 The Functions f, g, and E

In this section we define the important properties of the functions f, g, and E. For simplicity, we assume that f is commutative.

Property 1. *The encryption function E is secure against ciphertext-only attacks. Given a set of ciphertexts $c_0, c_1, \ldots, c_m$, where $c_i = E_k(p_i)$ is the encryption of an unknown plaintext p_i with an unknown key k, it is computationally infeasible to find any information about any plaintext p with non-negligible probability.*

This property prevents an attacker from discovering blinded keys by decrypting the ciphertexts that are broadcast to the group. It is necessary for the security of the system.

Property 2. *The function g is one-way; given $g(x)$, it is computationally infeasible to find x with any non-negligible probability.*

This property makes it impossible to find a node key k_x given the blinded key $k'_x = g(k_x)$. It is necessary for the security of the system.

Property 3. *Given a sequence $x_0, x_1, \ldots, x_{l-1}$, it is computationally infeasible to find k_l with any non-negligible probability, where $k_{i+1} = f(x_i, g(k_i))$, and k_0 is chosen at random, for any l less than some value l_{max}.*

This property makes it impossible for an evicted member, whose distance to the root is less than $l_{\max}$, to find any node key. We conjecture that it is sufficient for the security of the system. From the system invariant, a member at node y who is evicted knows the blinded keys of the sibling nodes along the path from y to the root (which we can label $x_0, x_1, \ldots$). This is the only knowledge about the state of the key tree after her eviction that she has, and because of the second property, it is insufficient to violate the security of the system. This property requires that the height of the key tree must be less than $l_{\max}$.

Property 3 prevents the complete set of evictees from finding any unblinded node keys through collusion, since the most recent evictee gains no useful knowledge from collusion. After the multiple eviction of an arbitrary set of members, the combined knowledge about the state of the key tree is the blinded keys of the CAT-siblings. The cryptanalytic problem that a colluding set of multiply-evicted members face is *more* difficult than that of a single evictee, since there is no sequence of blinded keys that extends from a leaf to the root.

4.2 Discussion

The cryptanalytic challenge of property 3 is similar to the problem of guessing $g^l(x)$ (that is, the l^{th} iterate of the function g on x), where x is chosen at random. This construction is used by the one-time password system [7] and the S/KEY protocol [5] that implements it. Other cryptosystems, such as the group key management method in section 2 of [3], rely on a similar construction. When l is sufficiently large, this construction is weak. If $g : 2^m \to 2^m$ is iterated $l = 2^{m/2}$ times, this construction has only about $m/2$ bits of security, since the set of possible outputs includes only about $2^{m/2}$ elements (with overwhelming probability) [4]. However, this construction appears to be secure when $l \ll 2^{m/2}$.

The tree construction has half as many random inputs (assuming a balanced tree) as it has function evaluations, while the iterated construction has only one input. This fact suggests that the tree construction used in OFT is more secure than the iterated construction mentioned above.

The theoretical implications of choosing an appropriate value of $l_{\max}$ is an interesting open question. Nevertheless, we conjecture that if Property 2 holds, then for $f(x, y) = x \oplus y$, Property 3 holds for $l_{\max} = 30$. This choice, which is less than the standard number of 100 iterations used in the S/KEY protocol, is sufficient to handle over one billion users.

5 Computational, Transmission, and Storage Requirements

In this section, we analyze and compare the time, space, and communication requirements of our new OFT method with those of the SKDC and LKH methods. Tables 1 and 2 summarize our comparisons, focusing on the following measures: broadcast size, unicast size, manager computation, maximum of all member computations, manager storage, and member storage. Although the three methods can be objectively compared by these quantitative comparison criteria, the decision of what method is best for a particular application must depend on the requirements of the application. In the rest of this section, we distill the most important differences among the three methods, as revealed by Tables 1 and 2.

Tables 1 and 2 give the complexities of the group initialization, key establishment, and re-keying tasks. These tasks are independent of the final step of carrying out secure group communications with an established key.

5.1 Terminology

Our analysis assumes that the trees used by LKH and OFT are binary; broadcast sizes increase with the branching ratio of the trees, so binary trees minimize the communication cost, and are a practical choice. All of the values neglect constant additive factors for readability. We express these values in terms of constants which represent basic requirements of f, g, and E. K is the size of a key in bits. C_E, C_r, and C_g are respectively the computational cost of one evaluation of the encryption function E, generating a key from a cryptographically secure random source, and of one evaluation of g. The number of members in the group is n.

In the LKH and OFT methods, the member 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. 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 $\lg n$. A rebalancing operation can be performed to restructure the tree so that its height is no greater than $\lg n + 2$.

5.2 Results

The broadcast size to re-key after an addition or eviction is n keys for SKDC, $2hK + h$ bits for LKH, and $hK + h$ for OFT. OFT achieves a smaller broadcast than LKH through its bottom-up approach. The extra factor of 2 in LKH comes from the fact that in LHK 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.

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 keys (there are h of the later). 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 keys which have changed. Nothing else needs to be broadcast or changed (the blinded keys 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 two interesting steps: (a) 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 keys. (b) 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 must be communicated.

The storage requirements of both LKH and OFT members are about $\lg n$ keys, while that of SKDC members is exactly 2 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 many practical applications entropy is a precious resource.

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

Initializing Group

	SKDC	LKH	OFT
Broadcast size (bits)	nK	$2nK + h$	$2nK + h$
Manager comp.	$n(C_E + C_r)$	$2n(C_E + C_r)$	$2n(C_E + C_g) + nC_r$
Max. member comp.	C_E	hC_E	hC_E

Adding a member

	SKDC	LKH	OFT
Broadcast size (bits)	$nK + \lg n$	$2hK + h$	$hK + h$
Unicast size	-	-	hK
Manager comp.	$nC_E + C_r$	$h(2C_E + C_r)$	$h(C_E + 2C_g) + C_r$
Max. member comp.	C_E	hC_E	$h(C_E + C_g)$

Adding l members

	SKDC	LKH	OFT
Broadcast size (bits)	$(n + l)K$	$2s_l K + lh$	$s_l K + lh$
Unicast size	-	-	lhK
Manager comp.	$(n + l)C_E + lC_r$	$C_E(2s_l - l) + C_r s_l$	$s_l C_E + 2C_g(s_l - l)C_r l$
Max. old member comp.	C_E	hC_E	$h(C_E + C_g)$

Evicting a member

	SKDC	LKH	OFT
Broadcast size (bits)	$nK + \lg n$	$2hK + h$	$hK + h$
Manager comp.	nC_E	$h(2C_E + C_r)$	$C_r + h(C_E + 2C_g)$
Max. member comp.	C_E	hC_E	$h(C_E + C_g)$

Evicting l members

	SKDC	LKH	OFT
Broadcast size (bits)	$(n - l)K$	$(2s_l - l)K + lh$	$s_l K + lh$
Manager comp.	$(n - l)C_E + C_r$	$(2s_l - l)(C_E + C_R)$	$s_l(C_E + 2C_g) + C_R$
Max. member comp.	C_E	hC_E	$h(C_E + C_g)$

Table 1: Comparison of the maximum computational and transmission requirements of three important group keying methods. These numbers are approximate. n is the number of members in the group, h is the height of the key tree, and s_l is the size of the CAT when l leaves change. K is the size of a key in bits. C_E, C_r, and C_g are respectively the computational cost of one evaluation of the encryption function E, generating a key from a cryptographically secure random source, and of one evaluation of g.

	SKDC	LKH	OFT
Manager storage	nK	$2nK$	$2nK$
Member storage	$2K$	hK	hK

Table 2: Storage requirements of the group manager and group members, for three important group keying methods.

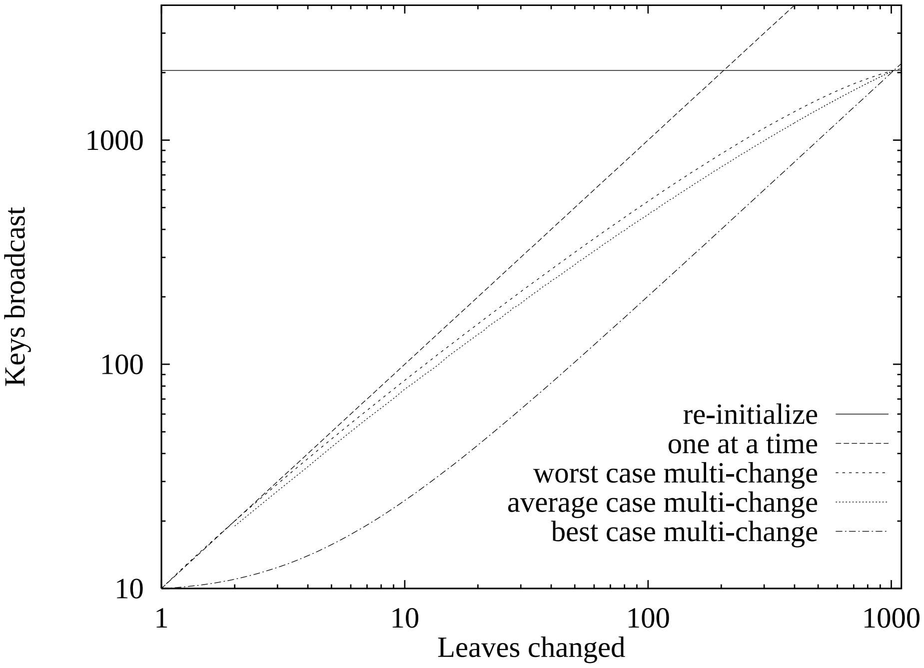

Figure 2: The number of keys broadcast by the OFT method to re-key after an addition or eviction, as a function of the number of changing leaves, in a group with 1024 members (and a balanced tree). The graph is drawn on a log-log scale, so that all data are legible. Multi-change refers to multiple eviction or addition.

5.3 Multiple Addition and Eviction

In the set eviction and set addition operations, the broadcast size and the computational effort depend on the size of the CAT. Upper and lower bounds on the size s_l of a CAT with l leaf nodes can be computed, allowing bounds to be put on the broadcast size and computational effort.

The minimum value of s_l occurs when the CAT is a size $2l-1$ subtree at the lowest level, attached to the root by a length $\lfloor \log_2(n/l) \rfloor$ path. The maximum value of s_l occurs when the CAT is a size $2l-1$ subtree, with l paths of length $\lfloor \log_2(n/l) \rfloor$ from its leaves downward. The size s thus satisfies

$$2l - 1 + \lfloor \log_2(n/l) \rfloor < s_l < 2l - 1 + l \lfloor \log_2(n/l) \rfloor, \qquad (2)$$

when n and l are powers of 2; the bounds are slightly less tight in the general case.

When l leaves are selected at random from a set of n, it is highly unlikely that they will be clustered together. The minimum value of s_l is attained only when the changing leaves are adjacent; for this reason, we expect the average value of s_l to be closer to its maximum. Figure 2 shows the maximum, average, and minimum values for s_l as a function of l. As expected, the average value is close to the maximum value; it is never more than 15% smaller.

The CAT size in the set addition operation can come close to achieving the lower bound, since the group manager can choose the locations of the new members to be adjacent.

In the set add and set evict operations, the broadcasts must communicate the changes in the topology of the CAT. If the members know the current topology of the tree, then this information is completely specified by a list of the leaves that are changing. When l leaves change in a group with n members, the identities of the leaves can be specified with $l(1 + \lfloor \log_2 n \rfloor)$ bits. Alternatively, the broadcast message can identify each blinded key with a node.

6 Composing Groups of Groups

We have made the unnecessary assumption that the members of a group are individuals; a group may be composed of both individuals and subgroups. When a subgroup A is added to a group G, a node in the key tree of G is split, and A is assigned to the right child of that node. The group manager of G communicates the leaf key of A's node encrypted with the group key k_A of A (in other words, OFT is used as the secure external channel, mentioned in Section 3.1, to communicate with A).

6.1 Unions and Intersections

New groups, which are the union or intersections of subgroups, can be created, while the subgroups retain their identities as subgroups (and their [sub]group keys).

To create a group key for the group $G \cup H$, create a new key tree with two leaves x and y; give k_x and k'_y to G encrypted with the group key k_G, and give k_y and k'_x to H encrypted with the group key k_H. The group key $k_{G \cup H}$ is given by $f(k'_x, k'_y)$, as above.

A set intersection operation assigns a subgroup to a node but does not give it the blinded node key of the sibling node. To create a group key for the group $G \cap H$, define the key $k_{G \cap H}$ as $f(k'_x, k'_y)$; give k_x to G encrypted with the group key k_G, and give k_y and to H encrypted with the group key k_H. Only the members of $G \cap H$ have both k'_G and k'_H, and so only they can compute $k_{G \cap H}$.

If the membership of G changes, then k_G also changes, and $k_{G \cup H}$ or $k_{G \cap H}$ must change as well. Importantly, our key management method allows efficient re-keying of composite groups: if the composite tree is balanced, then re-keying will require a broadcast which is logarithmic in the size of the total group.

With these simple rules, groups can be created out of arbitrary unions and intersections subgroups. This facilitates group management by providing an efficient means of composing groups in terms of other groups.

7 Implementation Notes

Although the implementation of the OFT method is straightforward, several important engineering decisions must be made. Among these decisions are the choice of f and g, the representation of the tree, time-space tradeoffs by each member involving whether or not to store unblinded ancestor keys, and the format of broadcasts by the manager.

We believe that the function g can be based on a cryptographic hash function such as MD5 [11] or SHA-1 [10]. It is possible that the root key does not need to be as large as the output size of the underlying function. For example, MD5 has a sixteen byte output, while DES keys are only seven bytes long. In this case, the MD5 output can be reduced by discarding data, as is done in by S/KEY, so that the node keys (and thus the broadcasts) are smaller.

The function f does not need to be one-way; it only needs to mix its inputs. This fact suggests that $f(x, y) = x \oplus y$ is a good choice.

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

In a full system, one might like to support a variety of additional operations. 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 later functionality be effected through a separate key repository mechanism that is independent of the key establishment algorithm.

8 Conclusions

We have presented and analyzed a new practical 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).

Unlike all previously proposed solutions based on information theory, public-key cryptography, hybrid approaches, or a single key distribution center, our OFT algorithm has communication, computation, and storage requirements which scale logarithmically with group size, for the add or evict operation. Each of the aforementioned approaches scale linearly or worse.

In comparison with the only other proposed hierarchical method—the Logical Key Hierarchy (LKH) [14]— our OFT algorithm reduces by half the number of bits broadcast by the manager per add or evict operation; the user time and space requirements of OFT and LKH are roughly comparable. For many applications, including multicasts, minimizing broadcast size is especially important.

Our security analysis of OFT raises some interesting questions about the security of function iterates, and that of bottom-up one-way function trees.

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

We would like to acknowledge contributions by several colleagues at Trusted Information Systems. We are very grateful to Michael V. Harding for significant contributions to the original algorithm and for discussing implementation strategies with us. Dennis K. Branstad, Principal Investigator of the Dynamic Cryptographic Context Management (DCCM) Project, suggested the problem and provided constructive recommendations and technical guidance. Thanks to Jay Turner for helpful comments, and to Sharon Osuna and Caroline Scace for editorial suggestions.

References

[1] Blundo, Carlo, Alfred de Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung" in *Advances in Cryptology: Proceedings of Crypto 92*, E. F. Brickell, ed., LNCS 740, Springer-Verlag (1992), 471–486.

[2] Burmester, Mike, and Yvo Desmedt, "A secure and efficient conference key distribution system" in *Advances in Cryptology: Proceedings of Eurocrypt 94*, A. De Santis, ed., LNCS 950, Springer-Verlag (1994), 275–286.

[3] Fiat, Amos, and Moni Naor, "Broadcast encryption" in *Advances in Cryptology: Proceedings of Crypto 93*, D. R. Stinson, ed., LNCS 773, Springer-Verlag (1993), 481–491.

[4] Flajolet, Philippe, and Andrew M. Odlyzko, "Random Mapping Statistics," *Advances in Cryptology: Eurocrypt '89 Proceedings*, J.J. Quisquater and J. Vandewalle, eds., LNCS 434, Springer-Verlag (1989), 330–354.

[5] Haller, Neil M., "The S/KEY one-time password system," *Proceedings of the Internet Society 1994 Symposium on Network and Distributed System Security*, Internet Society, (1994), 151–157.

[6] Harney, Hugh, Carl Muckenhirn, and Thomas Rivers, Group Key Management Protocol Architecture, *Request for comments (RFC) 2093*, Internet Engineering Task Force, (March 1997).

[7] Lamport, Leslie, "Password authentication with insecure communication," in *Communications of the ACM*, vol. 24, no. 11 (November 1981), 770–772.

[8] Menezes, Alfred J., Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press (Boca Raton, 1997).

[9] Merkle, Raph C., "Secrecy, authentication, and public-key cryptosystems," Technical Report No. 1979-1, Information Systems Laboratory, Stanford University (Palo Alto, CA, 1979).

[10] FIPS Publication 180-1, *Secure hash standard*, NIST, U.S. Department of Commerce, Washington, D.C. (April 1995).

[11] Rivest, Ronald L., "The MD5 Message-Digest Algorithm," Request for Comments (RFC) 1321 (1992).

[12] Steiner, Michael, Gene Tsudik, and Michael Waidner, "Diffie-Hellman key distribution extended to group communication," *Proceedings of the 3rd ACM Conference on Computer and Communications Security* (March 14–16, 1996). 7 pages.

[13] Steiner, M., G. Tsudik, and M. Waidner, "CLIQUES: A new approach to group key agreement," IBM Research Report RZ 2984 (# 93030), (December 12, 1997). 17 pages.

[14] Wallner, Debby M., Eric J. Harder, and Ryan C. Agee, "Key management for multicast: Issues and architectures," *Internet Draft* (Work in progress), Internet Engineering Task Force (July 1, 1997). 18 pages.

[15] Wong, Chung Kei, Mohammed G. Gouda, and Simon S. Lam, "Secure group communications using key graphs," Technical Report TR-97-23, Dept. of Computer Science, Univ. of Texas at Austin (July 28, 1997). 24 pages.

Appendix A: Pseudocode for OFT Algorithms

Algorithm 1. Manager-AddMember (T, M)
input: an OFT T, and a new member M
action: modify T, and broadcast M's blinded ancestor sibling keys to members

```
Begin
   x = SelectAdditionLeaf(T)
   OldM = Member(x)
   (a, b ) = Split(x)
   Assign(a, OldM); Assign(b, M);
   AssignNewKey(OldM); AssignNewKey(M);

   B = BuildListOfBlindedAncestorKeys(b)
   Broadcast("Add", NodeNumber(b), B)
   L = BuildListOfBlindedAncestorSiblingKeys(b)
   Unicast(L) to M
End
```

Note: Upon receipt of the broadcast, each member recomputes the shared group key by recomputing and storing her changed blinded ancestors.

Algorithm 2. Manager-EvictMember(T, M)
input: an OFT T, and a member M
action: modify T, and broadcast new blinded keys to members

```
Begin
   y  = Leaf(M)
   p  = Parent(y); s = sibling(y)

   if leaf?(s) then
      Assign(p, Member(s))
      x = s
   else
      p = s
      x = SelectLeaf(s)
   endif

   AssignNewKey( Member(x) )
   B  = BuildListOfBlindedAncestorKeys(x)
   Broadcast("Evict", NodeNumber(y), B)
End
```

Algorithm 3. Manager-MultiEvict(T, L)
input: an OFT T, and a list of members to evict
action: modify T, and broadcast new blinded keys to members

```
Begin
   C = ComputeEvictionCAT(T, L)
   B = BuildListOfBlindedAncestorKeysForCAT( C )
   Broadcast("MultiEvict", NodeNumbersOfMembers(L), B)
End
```