Efficient Group Key Management Protocol with One-Way Key Derivation

Efficient Group Key Management Protocol with One-Way Key Derivation

Jen-Chiun Lin∗ Feipei Lai† Hung-Chang Lee‡

Abstract

The growth of the Internet inspires lots of new network applications, and many of them are based on group communication models. Key trees are ideal for a group of users to efficiently and securely share a common secret key, the group key, which can encrypt transmitted data, or other session keys that protect group communication. An efficient group key management protocol is proposed for centralized dynamic groups, and one-way key derivation is integrated with key trees to reduce the communication overhead of rekeying operations. The server does not have to send new keys to those members who can derive the keys by themselves, and the total number of encrypted keys transmitted per rekeying operation becomes fewer. It is shown that the technique can be applied to both synchronous and asynchronous rekeying operations. The proposed protocol outperforms the other group key management protocols from our analysis and simulation, and is suitable for practical systems.

1 Introduction

The growth of the Internet inspires lots of new network applications, where a message originated from a user has to be sent to other users in the group. Typical group applications include distance education, video conferences, collaborative work, and distributed interactive simulation. Group communications can take advantages of a more efficient multicast service [7, 3], which is capable of sending a message to multiple destinations. Communication confidentiality is crucial for applications communicate via an open network environment, such as the Internet. A unique characteristic of group communications is that the parties participating in the communication can change with time, which complicates key management. While there is an abundance of solutions for traditional, point-to-point secure communications, Traditional point-to-point secure communications are usually not directly applicable to a secure group communication system due to poor scalability.

Key sharing schemes, such as key graphs [14], are proposed to let a group of users efficiently share a secret key, the group key, which can be used to encrypt the data transmitted among group members, or can be used to encrypt other session keys to secure group communication traffic. The process to securely distribute group keys to all members is called group key management protocol. Many such protocols rely on a trusted and secure server to manage the group key, as well as the membership of the group. These protocols must ensure that, for a user that does not belong to the group, it is computationally infeasible to derive any group key, and for a group member, it is computationally infeasible to derive group keys used before its participation and after its departure. The key server has to perform secure rekeying to send new group key to all members whenever the group key has to be changed to maintain communication secrecy [11]. Key trees and their variations are widely used in such secure group communications systems, since it is generally quite efficient to update shared group key based on such hierarchical schemes.

In this paper, one-way key derivation is integrated with key trees to make rekeying operations more efficient. The major benefit of this approach is that, while a new key is derivable by a subgroup of members themselves, it does not have to be encrypted and sent by the server, and therefore the bandwidth requirement is lessened. Our approach can be applied to both synchronous and asynchronous rekeying schemes, and new algorithms are proposed for the single join, single leave, and batch update operations. The communication costs of these operations are simulated with real network traffic, and the results show that the proposed protocol outperforms the other group key management protocols in our comparison.

The rest of the paper is organized as follows. Section 2 gives an overview of key trees and group key rekeying. Section 3 explains the concept of one-way key derivation, and the key update operations for both synchronous and asynchronous rekeying. Section 3 analyzes the performance of
several related group key management protocols, and their performance are evaluated and compared in our simulation. The last two sections give the related work and the conclusion.

2 Key Tree and Group Key Rekeying

For a centralized group communication system, the server is responsible to manage all the requests of group members, and to update the group key to maintain the communication secrecy of the group. Key trees [14], or logical key hierarchies (LKH) [6], are designed to support efficient and scalable group key rekeying for dynamic groups. A key tree is a hierarchical arrangement of a set of keys. Nodes of a key tree are called k-nodes, and an auxiliary key and an individual key are assigned to an internal k-node and an external k-node, respectively. Each external k-node is associated with a unique u-node, which represents a particular member in the group. An individual key is only shared by the server and the corresponding member, and an auxiliary key is shared by members that belong to the subtree rooted at the k-node of the auxiliary key. An individual key is usually established at the time a user joins the group, and it often remains valid until the user leaves. On the other hand, auxiliary keys are frequently updated and sent to members to keep group communication secrecy. The root k-node stores the group key, which is a special auxiliary key shared by all members.

Figure 1 shows how the server handles join and leave requests, and how the group key is updated and distributed to all members. Assume that there is a key server that manages a group with 8 members, and, before a new user $u_9$ joins the group, the server uses the key tree in Figure 1(a). Now if a new user $u_9$ sends a join request to the key server and joins the group, and is placed as a sibling of member $u_8$, the new key tree is as shown Figure 1(b). The server creates a new u-node for $u_9$, associates the u-node with a new k-node, and assigns new individual key $k_9$ to the k-node. Since the key tree is changed by the join operation, the server has to send new keys in the shaded k-nodes to the sharing members. The new keys can be securely distributed to members:

$$ s \rightarrow u_1 - u_6 : [k_{1-9} | k_{1-8}] $$
$$ s \rightarrow u_7, u_8 : [k_{7-9} | k_{7-8}] $$
$$ s \rightarrow u_9 : [k_{9-7} | k_{9-8}] $$

where $[k']_k$ denotes key $k'$ encrypted with key $k$, and $s$ denotes the server.

If later user $u_9$ wishes to leave the group, and suppose that before it leaves the group, the key tree is like Figure 1(b), the server removes the corresponding u-node, the associated k-node, and changes the key tree to Figure 1(a). Next, it generates new keys $k_{1-8}, k_{7-8}$, and sends them to the remaining members:

$$ s \rightarrow u_1 - u_3 : [k_{1-8} | k_{1-3}] $$
$$ s \rightarrow u_4 - u_6 : [k_{1-8} | k_{4-6}] $$
$$ s \rightarrow u_7 : [k_{7-8} | k_{7-8}] $$
$$ s \rightarrow u_8 : [k_{9-7} | k_{9-8}] $$

In practice, synchronous rekeying operations, such as single join and single leave operations that cause immediate group key rekeying after each request can incur much communication overhead. By queuing several requests and performing one rekeying for them, asynchronous rekeying can take advantage of the possible overlap of new keys for multiple join or leave requests, and, therefore, can reduce the communication cost. Li et al. [10] also suggested that asynchronous rekeying can alleviate the out-of-sync problem the synchronous rekeying scheme suffers. However, asynchronous rekeying has a drawback that it will enlarge the vulnerability window, though the security degradation is usually acceptable for better system performance.

Figure 1. Key trees (a) before $u_9$ joins (after $u_9$ leaves) (b) after $u_9$ joins (before $u_9$ leaves).
3 One-way key derivation protocol (OKD)

The basic idea behind one-way key derivation is simple. If a member can compute a new key the way the server does, the server does not have to send the new key to the member. Therefore, the rekeying bandwidth can be reduced. The method is integrated into ordinary single join operation, single leave operation, and batch update operation to make them more bandwidth efficient.

3.1 One-Way Key Derivation

A strong one-way key derivation function is crucial to the security of the protocol. The derivation function $f(\cdot)$ is used to generate new key values based on old key values. A good derivation function should have the following two properties.

Property 1 (One-way) Given $k$, it is easy to compute $f(k)$, but given $f(k)$, it is computationally infeasible to compute $k$.

Property 2 (Randomness) Given $k_0$, $k_1$, ..., $k_n$, it is computationally infeasible to compute $f(k)$, if it is computationally infeasible to compute $k$.

Satisfying these two properties make sure that the key values used to derive new keys will not be easily computed, and the generated keys are unpredictable without knowing the related information used to compute them. To avoid producing repetitive key values with the same input key value $k$, in some occasions, a non-zero salt value $K$ is used in

$$f(k \oplus K),$$

which is able to produce different values even with the same $k$ when $K$ varies, and $K$ should be chosen so that it is infeasible to deduce $k$ from it.

For practical systems, secure hash functions, such as SHA-1 [4] or SHA-2 [5], are good candidates of the key derivation function. For instance, suppose that 128-bit keys are used as auxiliary keys and individual keys, SHA-1 can be used to build a one-way key derivation function:

$$f(k) \equiv \text{SHA-1}(k) \mod 2^{128}.$$

3.2 Synchronous Rekeying Operations

The protocol supports single join and single leave operations to handle join and leave requests.

3.2.1 Single Join Operation

Each time a new member wishes to join the group, it sends a join request to the server. After receiving the request, the server verifies it, authenticates the identity of the new user, randomly generates a new individual key for the member, and sends it via a secure channel. The server then assigns a new $u$-node to represent the new member, and creates a new $k$-node to hold the new individual key. The $k$-node is inserted in the end of one of the shortest paths of the key tree.

![Figure 2. Key derivation (single join).](image)

After the new $k$-node is inserted into the key tree, all auxiliary keys of the internal $k$-nodes on the path from the $k$-node of the new member to the root must be updated. Figure 2 shows how these new keys are computed. For every old $k$-node that already has an old auxiliary key $k_0$, the new auxiliary key is computed by

$$k' = f(k_0).$$

It is possible that a new $k$-node is created due to the last internal $k$-node being full. As shown in Figure 2(b), the new value is computed by

$$k' = f(k_1 \oplus k_g),$$

where $k_1$ is the individual key value of an old member, and $k_g$ is the old group key value. The derivation is always possible because when a new $k$-node is created, there must be an old external $k$-node associated with an old member moved down to become a child of the newly created $k$-node. The victim member also already knows both $k_1$ and $k_g$. The salt value $k_g$ is to ensure that $k'_0$ is different even when the same individual key is used in derivation, since $k_g$ will be different each time. After the new key values are computed, the server encrypts these new keys and sends them to all members via multicast or unicast channels.

We use Figure 1 to explain the join operation of new member $u_9$. A new $k$-node $x_9$ is created to hold $u_9$’s individual key $k_9$. Auxiliary keys of $k$-nodes $x_1$ and $x_7$ on the path are to be updated. According to Figure 2(a), new key values are

$$k_{1-9} = f(k_{1-8}), \quad k_{7-9} = f(k_{7-8}).$$
After the new keys are computed, the server continues to send the rekeying data to group members. The server only has to send the information about the position of the new k-node of the new member to notify all old members about the updated path. The server only encrypted the new keys for the new member who cannot derive the keys:

\[ s \rightarrow u_9 : \left[ k_{1-9} | k_{7-9} \right] \left| k_9 \right] \]

Comparing to the example in Section 2, it can be observed that the rekeying process for old members are greatly simplified by one-way key derivations. The bandwidth is also saved, since no keys are encrypted and sent to old members.

### 3.2.2 Single Leave Operation

Usually there are two situations that the leave operation will be triggered. Normally, when a group member wishes to leave the group, it sends a leave request to the server. After receiving the request, the server has to verify it to ensure that it really comes from the member. If the verification passes, the server can continue to finish the leave operation. A member is also possible to be expelled from the group by the server because of network failure, member’s misbehavior, or other reasons. The server then removes the u-node of the member and the k-node associated with the member from the key tree.

![Figure 3. Key derivation (single leave).](image)

After the k-node of the leave member is removed from the key tree, all auxiliary keys on the path from the position of the removed k-node to the root have to be updated. Figure 3 shows the one-way key derivation method used in this operation. Since all these updated k-nodes are old k-nodes, they can be computed by

\[ k'_0 = f(k_1 \oplus k_0), \]

where \( k_1 \) is one of the auxiliary keys of those k-nodes that is not on the leave path. The old auxiliary key \( k_0 \) is used as a salt value, and this will not cause security problem because only members knowing \( k_1 \) can compute the new key. This derivation will only benefit a subset of all the old members who owns \( k_1 \), and how to select the k-node \( k_1 \) depends on the policy of the server. Finally, these new keys are encrypted and sent to all members through multicast channels.

We use Figure 1 as an example to explain the leave operation of member \( u_9 \). After \( u_9 \) left the group, the associated k-node \( x_9 \) is no longer needed and is removed. The auxiliary keys of k-nodes \( x_{1-9} \) and \( x_{7-9} \) have to be updated. Suppose \( k_{1-3} \) and \( k_7 \) are chosen by the server to compute \( k_{1-8} \) and \( k_{7-8} \), respectively, it follows that

\[ k_{1-8} = f(k_{1-3} \oplus k_{1-9}), \quad k_{7-8} = f(k_7 \oplus k_{7-9}), \]

where \( k_{1-9} \) and \( k_{7-9} \) are salt values. Unlike the join operation, not all old members can derive the new keys, and the server has to encrypt and send these keys to members who cannot compute them:

\[ s \rightarrow u_4 - u_6 : \left[ k_{1-8} | k_{4-6} \right] \quad s \rightarrow u_7 : \left[ k_{1-8} | k_{7-8} \right] \quad s \rightarrow u_8 : \left[ k_{1-8} | k_{7-8} \right] \left| k_{7-8} \right] \]

Comparing to the example in Section 2, the bandwidth overhead of the rekeying operation is reduced using one-way key derivation.

### 3.3 Asynchronous Rekeying Operations

Three operations, the join operation, the leave operation, and batch update operation are supported. Among them only the batch update operation performs the actual rekeying. The period of time between two successive batch update operations is called the batch interval.

#### 3.3.1 Join Operation

This operation is similar to the single join operation, except that there is no key updates. The server verifies the join request is authentic, generates a new individual key, and securely sends it to the new member. The member is then queued to wait for the rekeying message by the next batch update operation.

#### 3.3.2 Leave operation

The conditions to trigger a leave operation are similar to the single leave operation. The server first checks if the leave member is also a new member in the same interval. If it is, the join and the leave requests cancel out each other. Otherwise, the request is queued, and will be processed in the next batch update operation.

#### 3.3.3 Batch Update Operation

In the operation, the server processes all queued join and leave requests in the interval, generates new keys, and sends them to group members. Our batch update algorithm is an enhanced version of the one in [16]. First of all, the server
removes all the u-nodes of leave members and the k-nodes containing their individual keys. Meanwhile, all k-nodes on the leave paths are marked LEAVE. If there are new members, their k-nodes are either inserted in the end of one of the shortest paths, or distributed equally at the places of the k-nodes of the leave members, if there is any. During the insertion process, all the old k-nodes on the join path are marked JOIN, and the newly created k-nodes marked NEW. It should be noted that these flags are not mutually exclusive. For instance, if a k-node is marked JOIN and LEAVE, its new auxiliary key will be used by old and new members.

![Figure 4. Key derivation (batch update).](image)

After the key tree is adjusted and marked, the server generates new auxiliary keys for all marked k-nodes as shown in Figure 4. Case (a): If the k-node $x_0$ is marked LEAVE, whether or not it is marked JOIN does not matter how the new key is generated. The server selects the auxiliary key $k_1$ of one of its child k-nodes as the derivation key, and computes the new key value

$$k'_0 = f(k_1 \oplus k_0).$$

Case (b): The k-node is marked JOIN but not LEAVE, and its old auxiliary key value will be used as the derivation key. The server computes

$$k'_0 = f(k_0).$$

Case (c): The k-node is a newly created k-node. If all of its child k-nodes are new k-nodes, the new key is randomly generated by the server. Otherwise, the server uses the individual key $k_1$ of the old member to compute the new auxiliary key by

$$k'_0 = f(k_1 \oplus k_g),$$

where $k_g$ is the old group key not known to the new members. The server must compute these new keys in a bottom-up fashion, since the key value $k_1$ can be a newly computed auxiliary key sometimes. In case (a), it is possible that all children of the k-node are marked LEAVE, and newly derived key of k-node $x_1$ rather than its old key value should be used, so that the leave members are not able to compute them. The server encrypts the keys and uses unicasts or multicasts to send them to members.

![Figure 5. A batch update example.](image)

The algorithm can be demonstrated by the example in Figure 5. Assume that, in the rekeying interval, member $u_9$ leaves, member $u_{10}$ joins. K-nodes $x_4$ and $x_9$ are removed, and the new k-node $x_{10}$ is inserted at the original position of $x_9$. The new keys are

$$k_{5,6} = f(k_5 \oplus k_{4-6}),$$
$$k_{7,8,10} = f(k_7 \oplus k_{7-9}),$$
$$k_{1-3,5-8,10} = f(k_{1-3} \oplus k_{1-9}),$$

where auxiliary keys of k-nodes $x_{1-3}$, $x_{5}$ and $x_{7}$ are chosen for new key derivations. If the server somehow chooses the auxiliary key of $x_{4-6}$ to compute the new key of $x_{1-9}$, according to key derivation order, the new key value $k_{1-3,5-8,10}$ will be $f(k_{5,6} \oplus k_{1-9})$ rather than $f(k_{4-6} \oplus k_{1-9})$. The bottom-up key generation approach guarantees that leave members cannot compute any new auxiliary keys. These new keys are then sent to all members:

$$s \rightarrow u_5: [k_{1-3,5-8,10}]_{k_{5,6}}$$
$$s \rightarrow u_6: [k_{1-3,5-8,10}]_{k_{5,6}}[k_{5,6}]_{k_6}$$
$$s \rightarrow u_7: [k_{1-3,5-8,10}]_{k_{7,8,10}}$$
$$s \rightarrow u_8: [k_{1-3,5-8,10}]_{k_{7,8,10}}[k_{7,8,10}]_{k_8}$$
$$s \rightarrow u_{10}: [k_{1-3,5-8,10}]_{k_{7,8,10}}[k_{7,8,10}]_{k_{10}}$$

Comparing to three separated join operations and leave operations, the advantage is that the overlapped keys of k-nodes $x_{1-9}$ and $x_{7-9}$ are saved.

3.4 Security Analysis

Comparing the rekeying operations of our protocol to those of LKH, the server running OKD will not randomly generate a new key value, if some kind of key derivation can be applied. The ways the old auxiliary keys are chosen to be updated and the structure of the key tree is adjusted in
each rekeying operation of OKD are the same as those in the corresponding operation of LKH. OKD basically improves LKH by removing unnecessary encryptions without altering other aspects of LKH. Therefore, with a strong one-way key derivation function, OKD is as secure as LKH.

4 Performance Comparisons

Three different protocols, including LKH [6], OFT [13], and ELK [12], are used to compare the performance of our protocol OKD. Their performance is analyzed, and the result is also verified in our simulation.

4.1 Analysis

Table 1 shows the communication costs of LKH, OFT, ELK, and OKD. For single join operations, OKD and ELK do not have to send any new keys and are the best choices. For single leave operations, OKD and OFT need half the bandwidth requirement of LKH and ELK. Therefore, considering synchronous rekeying operations with binary key trees, it is clear that our protocol is the overall best choice.

Considering asynchronous rekeying operations with binary key trees, batch update operations offer more flexibility than multiple join and multiple leave operations. They are also more bandwidth efficient, because more overlapped keys can be saved. Protocol OKD is the most bandwidth efficient, because the costs to update the keys for k-nodes marked LEAVE ($s_L$) and k-nodes marked JOIN ($s_J$) are the lowest.

LKH with 4-ary key trees is the most efficient according to [14, 10], but this is not necessary true for our protocol. We use the following simple scenario to show the idea. According to the table, the multicast cost of performing one single join and one single leave operation of OKD is approximately

$$C(d) = K(d - 1) \log_d n,$$

and it follows that

$$C'(d) = K \log_d n(1 - (d - 1)/(d \ln d)).$$

Let $C'(d) = 0$, we find that $d$ is between 2 and 3. The result suggests that, if the group size remains steady with time when members join and leave, binary key trees and ternary key trees are very likely the best tree structures for OKD. To choose the best tree structure, we rely on simulation to determine the best degree of the key tree.

4.2 Simulation

In this section, we present the simulation results of different protocols in Figure 6. To compute the asynchronous rekeying costs, the server runs a periodic update scheme with 300-second rekeying interval. For OFT and ELK, the batch update operation is simulated by a multiple leave operation for all leave members and a multiple join operation for all join members in the interval. For LKH and OKD, key trees of degree two to five are evaluated. The numbers attached to LKH or OKD indicate the degrees of key trees. The data used in our simulation were collected by Mlisten in the MBone [1], which is a virtual network overlaying the Internet. The MBone served as a testbed for the development of multicast protocols and group conference tools. The data came from the real audio sessions in the MBone from November 18 to December 9, 1996, and we choose three most durable sessions as the input sequences of join and leave events in the simulation. The communication costs are measured by the numbers of keys transmitted, and the numbers are normalized to the number of keys multicast to new members (join unicast). The join unicast, join multicast, and bulk multicast are also measured. Symbol $n$ denotes the group size, $j$ the number of join members, $l$ the number of leave members. The length of a path of the key tree is denoted $h$. In particular, $h_2$ stands for binary key trees, and $h_d$ for $d$-ary key trees. Symbols $s_J$, $s_N$, $s_L$ are the numbers of k-nodes that are marked LEAVE, marked JOIN only, and marked NEW only, respectively. $K$ denotes the key length.

<table>
<thead>
<tr>
<th></th>
<th>LKH</th>
<th>OFT</th>
<th>ELK</th>
<th>OKD</th>
</tr>
</thead>
<tbody>
<tr>
<td>single join</td>
<td>multicast $2h_JK$</td>
<td>$b_2K$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>unicast $h_JK$</td>
<td>$b_2K$</td>
<td>$h_2K$</td>
<td>$h_4K$</td>
</tr>
<tr>
<td>single leave</td>
<td>multicast $dh_JK$</td>
<td>$b_2K$</td>
<td>$2h_2K$</td>
<td>$(d - 1)h_4K$</td>
</tr>
<tr>
<td>multiple join</td>
<td>multicast n.a.</td>
<td>$(h_2 + 1)K + h_2$</td>
<td>0</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>unicast $jh_2K$</td>
<td>$jh_2K$</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>multicast all</td>
<td>$(2s_J + 2j + 1)K$</td>
<td>$(s_J + 2j - 1)K$</td>
<td>n.a.</td>
</tr>
<tr>
<td>multiple leave</td>
<td>multicast n.a.</td>
<td>$(s_L + l)K$</td>
<td>$2s_JK$</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>unicast $jh_4K$</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>multicast all</td>
<td>$(ds_L + 2s_J + s_N + j)K$</td>
<td>$((d - 1)s_L + s_J + s_N + j)K$</td>
<td>n.a.</td>
</tr>
</tbody>
</table>

a Only the bandwidth to deliver the new keys are considered. Symbol $n$ denotes the group size, $j$ the number of join members, $l$ the number of leave members. The length of a path of the key tree is denoted $h$. In particular, $h_2$ stands for binary key trees, and $h_d$ for $d$-ary key trees. Symbols $s_J$, $s_N$, $s_L$ are the numbers of k-nodes that are marked LEAVE, marked JOIN only, and marked NEW only, respectively. $K$ denotes the key length.
For asynchronous rekeying operations, we also evaluated the communication costs for multicast only scheme (multicast all), where the server uses multicasts to deliver new keys to all new and old members. The join multicast all, and leave multicast all costs represent the multicast bandwidth overhead due to join requests, and multicast bandwidth overhead due to leave requests, respectively.

For synchronous rekeying operations, key trees of higher degrees will have lower unicast costs, because they will have a shorter average path length in general. For binary key trees, it is shown that the OFT and OKD2 introduce only half the numbers of updated keys of LKH2 in leave operations, and ELK and OKD2 do not have to transmit any key to old members in join operations. Among the four protocols, OKD2 has the best performance, since it has the lowest multicast overhead.

Depending on different sessions and protocols, the bandwidth reduction of asynchronous rekeying can be more than 70 percent. For all three sessions, about 41 percent to 58 percent of the join requests are cancelled by corresponding leave requests in a same interval. The join multicast cost of asynchronous rekeying is much lower than that of synchronous rekeying, since join paths usually overlap leave paths in an interval, and thus the overlap keys of join requests are saved. The result confirms that our protocol is best suited for binary key trees, for both the mixed uni-
5 Related Work

There has been a lot of researches on secure group communications. One of the early work that does not rely on special cryptographic routines is Iolus [11], which uses sub-groups for secure message delivery. Key graphs and key trees are proposed by Wong [14], and key trees have become a very popular approach of group key management. Logical key hierarchy, LKH [6], is a similar approach to arrange keys into a tree structure.

Sherman and McGrew [13] proposed one-way function trees for efficient key generation. Their work is subject to a particular kind of collusion attack [8], and an improvement is done by Ku in [9]. ELK [12] uses key derivations based on pseudo-random functions, and the authors also implement recovery mechanisms in their algorithms. LHK+ [2] uses one-way functions to improve the performance of join operations of LKH.

Keystone [15] uses forward error correction (FEC) to provide reliable group key delivery. Li et al. [10] gave a detailed discussion about batch rekeying for LKH. To optimize the performance of rekeying multicast, Yang and Li designed a R-BFS algorithm to take advantage of the cluster property of keys in multicast packets, and improve the performance of FEC multicast transmission [16]. In [17], a protocol supporting efficient packet encoding scheme was proposed, and parity packets are added to provide proactive error correction.

6 Conclusion

We present a one-way key derivation method to reduce the communication overhead of rekeying operations of centralized secure group communication systems with key trees. With one-way key derivation, the server does not have to encrypt and transmit new keys to members who can derive the keys by themselves. With strong one-way key derivation and encryption algorithms, OKD is as secure as LKH. The analysis and simulation show that it is the most bandwidth efficient protocol compared to LKH, OFT, and ELK, and binary key trees are overall of the best tree structure for OKD.

Acknowledgment

The authors would like to thank Professor K. C. Almeroth for the audio session data collected in the MBone.

References


