Reducing the Cost of the Critical Path in Secure Multicast for Dynamic Groups[1]

Sandeep S. Kulkarni Bezawada Bruhadeshwar
Department of Computer Science and Engineering
Michigan State University
East Lansing MI 48824 USA

Abstract

In this paper, we focus on the problem of secure multicast in dynamic groups. In this problem, a group of users communicate using a shared key. Due to the dynamic nature of these groups, to preserve secrecy, it is necessary to change the group key whenever the group membership changes. While the group key is being changed, the group communication needs to be interrupted until the rekeying is complete. This interruption is especially necessary if the rekeying is done because a user has left (or is removed). We split the rekeying cost into two parts: the cost of the critical path –where each user receives the new group key, and the cost of the non-critical path –where each user receives any other keys that it needs to obtain. We present two algorithms that show the trade off between the cost of the critical path and the cost of the non-critical path. We also compare our algorithms to previous algorithms and show that our algorithms reduce the cost of the critical path while keeping the total cost manageable.

Keywords : Group Communication, Multicast, Critical Path, Shared Keys

1 Introduction

Multicast is a necessary and efficient form of group communication. Many new applications, especially those that require the users to pay for the data or those where certain data needs to be shared with several authorized users, are group oriented. In these applications, there is a need to secure the data from unauthorized users, and hence, encryption of the data is necessary.

One way to prevent access by unauthorized users is to use the simple group key management protocol [1], GKMP. In this protocol, data is encrypted using a cryptographic key which is known only to the participating users. In GKMP, a group controller generates and distributes the cryptographic group key –using the personal key of each user– to all the users in the group. All communication is encrypted using this group key. Although GKMP has been successfully implemented in many applications, it lacks scalability when the group is large and dynamic. If users can join and leave the group, the controller needs to modify the group key to reflect the current membership of the group. For example, when a user joins (respectively, leaves) a group, the group key needs to be changed so that the joining (respectively, leaving) user cannot access the past (respectively, future) communication.

[1]Email: sandeep@cse.msu.edu, bezawada@cse.msu.edu, Web: `http://www.cse.msu.edu/~{sandeep,bezawada}`, Tel: +1-517-355-2387, Fax: 1-517-432-1061

This work is partially sponsored by NSF CAREER 0092724, ONR grant N00014-01-1-0744, DARPA contract F33615-01-C-1901, and a grant from Michigan State University.

Although the tasks involved when the user joins are straightforward in GKMP, the cost of rekeying when a user leaves is high; the group controller must send the new group key to each user separately using a secure channel.

One approach to deal with the cost of the leave operation is to use complementary variables technique [2]. In this technique, the controller assigns a unique identifying number, $u_1, u_2, .., u_n$, to each of the users. It also generates the keys $c_1, c_2, .., c_n$. User u_i gets all the keys except c_i. Now, if some user, say u_i, leaves, the new group key can be distributed easily using the key c_i. Although the group key can be changed easily in the complementary variable technique, there is an extra overhead incurred while changing the keys $c_1, c_2, .., c_{i-1}, c_{i+1}, .., c_n$. If these keys are not changed then two users that leave the group can collude to obtain the new group key.

The above discussion suggests that the cost of rekeying can be split into two parts: the cost to change the group key and the cost to change the other keys that the users need to maintain. The group communication can proceed when each user has the new group key. Thus, the first part, the cost to change the group key, is on the critical path to restoring the group communication after a user has left. By contrast, the other keys could be changed in the *background*, i.e., after the group communication is restored.

The cost of the critical path is important for the leave operation only; for the join operation, the cost of the critical path is always two. The group controller needs to send the new group key twice; once by encrypting it with the old group key and once by encrypting it with the personal key of the joining user. However, if a user has left the group (or if it is forced to leave the group) and we want to ensure that the leaving user cannot decrypt any future messages sent to the group then the cost of the critical path is crucial in restoring the group communication.

In this paper, we focus on the tradeoff between the cost of the critical path and the total cost involved in rekeying when a user leaves. The cost of the critical path in the complementary variable technique is $O(1)$. However, the total cost in this technique involves sending $O(n)$ keys to $O(n)$ users and requires $O(n^2)$ encryptions. Thus, the total cost of rekeying is $O(n^2)$.

Our approach is based on the complementary variable technique [2] and the logical key hierarchy [3]. In the logical key hierarchy, the keys are arranged in a tree, the users are at leaf nodes in this tree and each user receives all the keys on the path towards the root. In [3], the critical cost of the rekeying is $(d-1)h$ where d is the degree of a node and h is the height of the tree. We present two algorithms for combining the schemes in [2, 3] and show how the cost of the critical

path can be reduced while slightly increasing the total cost.

Organization of the paper. This paper is organized as follows. In Section 2, we describe our notations and show how the cost of rekeying is calculated. Then, in Sections 3 and 4, we present our algorithms. In Section 5, we compare the cost of the critical path for our solutions to that in [2, 3]. Finally, in Section 6, we give concluding remarks and point out future directions.

2 Notations

Similar to the logical key hierarchy scheme [3], in our algorithms, we arrange the users and the keys in a tree. The leaf nodes in this tree correspond to the users in the group (cf. Figure 1). We number the users and the keys according to their location in the tree. For example, in a tree of height h, the user $u_{i_1 i_2 .. i_h}$ denotes the user obtained by taking the i_1^{th} child at level 1, i_2^{th} child at level 2, and so on. The keys are also numbered likewise. Since we combine the logical key hierarchy scheme and the complementary variable scheme, each internal node may be associated with a key related to one or both schemes. We use $k_{i_1 i_2 .. i_l}$ to denote the key associated with the logical key hierarchy at node $i_1 i_2 .. i_l$. And, we use $c_{i_1 i_2 .. i_l}$ to denote the key associated with the complementary key variables at node $i_1 i_2 .. i_l$. Also, each user $u_{i_1 i_2 .. i_h}$ has a private key $k_{u_{i_1 i_2 .. i_h}}$ that is known only to the user and the group controller. If necessary, such a key could be established using public keys. For our algorithms, the approach used to establish this key is irrelevant. Finally, we use $dummy$ (with appropriate subscripts) to denote a temporary key; such a key is discarded after rekeying is complete.

As an illustration, in Figure 1, we have shown part of a tree of height 3; the logical key hierarchy is used at the first level and the complementary keys are used at the two levels below.

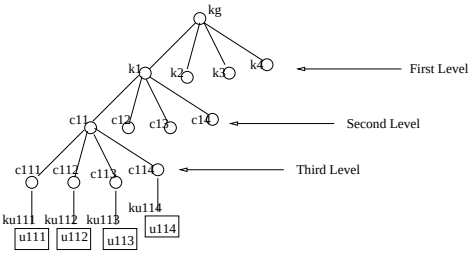

Figure 1. Representation of Users and Keys

The encryption of messages m_1, m_2,.. by key k_i is denoted by $k_i(m_1, m_2 ..)$. In our algorithms, the controller needs to send the necessary keys to appropriate users. Each transmitted key is encrypted by a key that is known (only) to the intended recipients. We use $k(c)$ denote that key c is encrypted using the key k and, hence, users that have the key k can obtain c.

The adversary (anyone outside the group) can listen to all messages sent over the network. Hence, for simplicity, we assume that all communication is broadcast in nature and, hence, we do not explicitly identify the intended recipients of a message.

We compute the complexity of our algorithms in terms of the number of encryptions and in terms of the number of messages. We assume that in one message, it is possible to send multiple keys if all keys being transmitted are encrypted by the same key. Thus, if the algorithm requires that the keys $k_1, k_2, .., k_m$ be transmitted by encrypting them with key k, only 1 message, $k(k_1, k_2, .., k_m)$, is sent. However, for this message, the number of encryptions is m.

3 First Algorithm: Partial Logical and Complementary Keys

In this algorithm, we partition the tree so that at the first h_1 levels the logical key hierarchy is used and at the next h_2 levels the complementary keys are used. To present the algorithm, we proceed as follows. First, we state the steady-state configuration of our algorithm. Then, we present the tasks involved when a new user joins and the tasks involved when a user leaves. After all the tasks for join (respectively, leave) are performed, the algorithm reaches another steady state configuration.

Steady state configuration. Given a tree of height h, we partition it into two parts; in the first h_1 levels, we use the logical key hierarchy and then in the remaining h_2 (=$h - h_1$) levels, we use the complementary key hierarchy. For the part of the tree where we use logical key hierarchy, the user obtains the keys on the path to the root. For this part, we let d_1 be the (maximum number of) children for each internal node in the tree. For the part where we use the complementary keys, the user gets the keys associated with the siblings of its ancestors. It does not get the keys associated with its ancestors. For this part, we let d_2 be the (maximum number of) children for an internal node. As an illustration, in Figure 1, u_{112} gets the keys k_g, k_1, $c_{12}, c_{13}, c_{14}, c_{111}, c_{113}$ and c_{114}. More generally, in a tree of height h, the user $u_{i_1 i_2 .. i_{h_1} i_{h_1+1} .. i_h}$ has the following keys,

- Group key k_g
- Logical keys $k_{i_1}, k_{i_1 i_2}, .., k_{i_1 i_2 .. i_{h_1}}$, and
- Complementary keys
  - $c_{i_1 i_2 .. i_{h_1} l}$, where $l \neq i_{h_1+1}$
  - $c_{i_1 i_2 .. i_{h_1} i_{h_1+1} l}$, where $l \neq i_{h_1+2}$,
  - ..,
  - $c_{i_1 i_2 .. i_{h-1} l}$, where $l \neq i_h$

For simplicity, we assume that the above tree is complete, i.e., at levels $0..(h_1 - 1)$, each node has d_1 children. And, at the remaining levels, each node has d_2 children. However, the number of users may be less than the total leaves in this keytree. Also, for simplicity, we assume that the height of the tree does not change when a user joins/leaves. We would, however, like to note that our algorithms can be easily modified to deal with the case where these conditions are not satisfied.

Join. When a new user joins the group, the cost of the critical path is always two; the controller needs to send $k_g(k'_g)$ and $k_u(k'_g)$ where k_g is the old group key, k'_g is the new group key and k_u is the key of the joining user. The distribution of other keys is also similar; if key k is changed and the new key is k', the controller needs to send $k(k')$ and $k_u(k')$. Thus, it is easy to see that if a user gets x keys then the number of encryptions for the join operation is $2x$ and

the number of messages is $x+1$ (one message for the joining user and one message each for the key that needs to be changed).

Leave. To illustrate the leave, we first consider the example in Figure 1. Consider the case where the user u_{113} leaves the group. To preserve forward secrecy, the group controller generates a new group key and distributes it to the remaining users in the group. In addition, the controller generates new values for all keys known to u_{113} and other users in the group. The set of keys held by u_{113} are $c_{111}, c_{112}, c_{114}, c_{12}, c_{13}, c_{14}, k_1$ and k_g. Let the new keys generated by the controller be $c'_{111}, c'_{112}, c'_{114}, c'_{12}, c'_{13}, c'_{14}, k'_1$ and k'_g respectively.

As mentioned in the Introduction, the tasks involved in communicating the new group key, k'_g, are on the critical path whereas the tasks involved in changing other keys are not on the critical path. Now, we discuss the tasks involved in both paths.

Critical Path. In order to restore the group communication, all the remaining users need to know the new group key. To distribute the new group key, for the first level where logical keys are used, the controller sends $k_2(k'_g), k_3(k'_g)$ and $k_4(k'_g)$. For the remaining two levels, the controller uses the keys associated with the ancestors of the leaving user u_{113}, i.e, the controller sends $c_{113}(k'_g), c_{11}(k'_g)$. Now, we generalize the above example for a more general case as follows.

Critical path: After user $u_{i_1 i_2 .. i_h}$ leaves
// For the first h_1 levels
for $(j=1; j \le h_1; j^{++})$
for $(l=1; l \le d_1; l^{++})$
if $(l \neq i_j)$ send $k_{i_1 i_2 .. i_{j-1} l}(k'_g)$

// For the next h_2 levels
for $(j=h_1+1; j \le h; j^{++})$ send $c_{i_1 i_2 .. i_j}(k'_g)$

In the above algorithm, for first h_1 levels, we use the approach in [3]; we consider the ancestor of the leaving user at level l. Then, we use the keys associated with the siblings of that ancestor to send the group key. Thus, at each level, (d_1-1) encryptions (respectively, messages) are used. Hence, the total number of encryptions (respectively, messages) for first h_1 levels is $(d_1-1)h_1$. Then, at the last h_2 levels, we use the complementary keys associated with the ancestor of the leaving user. Thus, h_2 encryptions (respectively, messages) are used. Thus, we observe:

Observation 3.1 The number of encryptions (respectively, messages) for the critical path is $(d_1-1)h_1 + h_2$. □

Non-critical path. As in the case of the critical path, we first show how the algorithm behaves in the example in Figure 1. For users at the lowest level, i.e., for the siblings of u_{113}, the controller uses the individual keys of the users to distribute the shared keys whose values are changed at that level. In other words, for keys $(c'_{111}, c'_{112}, c'_{114})$, the controller sends the following messages.

- $k_{u_{111}}(c'_{112}, c'_{114})$, $k_{u_{112}}(c'_{111}, c'_{114})$, $k_{u_{114}}(c'_{111}, c'_{112})$

At the second level, the controller needs to send c'_{12}, c'_{13} and c'_{14}. The key c'_{12} needs to be sent only to the descendants of c_{11}, c_{13}, c_{14}. For the descendants of c_{11}, the controller uses c_{113} to send c'_{12}, as c_{113} is not available with u_{113}. For the descendants of c_{13}, we proceed as follows. First, we send a dummy key using c_{131} and c_{132}. Since all descendants of c_{13} have either c_{131}, c_{132} or both, all descendants of c_{13} can obtain this dummy key. Then, we use this dummy key to send c'_{12} (as well as c'_{14}). Thus, the controller sends the following messages.[2]

- $c_{113}(c'_{12}, c'_{13}, c'_{14})$
- $c_{121}(dummy_{12}), c_{122}(dummy_{12}), dummy_{12}(c'_{13}, c'_{14})$
- $c_{131}(dummy_{13}), c_{132}(dummy_{13}), dummy_{13}(c'_{12}, c'_{14})$
- $c_{141}(dummy_{14}), c_{142}(dummy_{14}), dummy_{14}(c'_{12}, c'_{13})$

At the first level, the key k'_1 is changed. To distribute this key to all the users who share it, the controller can use the complementary keys, c_{11} and c'_{12}. More specifically, for k'_1, the controller sends the following messages.

- $c_{11}(k'_1)$, $c'_{12}(k'_1)$

We now generalize the above approach for the case where the number of users is arbitrary. The algorithm works from the bottom of the tree and moves upward changing the necessary keys. At the bottom level, individual user keys are used to transmit the complementary keys. Then, we change the complementary keys at levels $h-1$ to h_1+1 . As described above, for these levels, we first send a dummy key and use this dummy key to send the complementary keys. By sending this dummy key first, we reduce the number of encryptions required; one dummy key is used to distribute several complementary keys. Finally, after the distribution of the complementary keys, these keys are used to distribute the new logical keys. Hence, the algorithm is as follows. (In this algorithm, if key c has not changed then, for simplicity, we let c' be the same as c.)

Non-critical path: After user $u_{i_1 i_2 .. i_h}$ leaves
Changing keys at the lowest level
for $(l=1; l \le d_2; l=l+1)$
for $(k=1; k \le d_2; k=k+1)$
if$(k \neq l \wedge k \neq i_h \wedge l \neq i_h)$ send $k_{u_{i_1 i_2 .. i_{h-1} l}}(c'_{i_1 i_2 .. i_{h-1} k})$

Changing other complementary keys
for $(j=h-1; j>h_1; j=j-1)$
for $(l=1; l \le d_2; l=l+1)$
create $dummy_{i_1 i_2 .. i_{j-1} l}$
// This key is sent to the subtree rooted at $c_{i_1 i_2 .. i_{j-1} l}$
send $c'_{i_1 i_2 .. i_{j-1} l1}(dummy_{i_1 i_2 .. i_{j-1} l}))$
send $c'_{i_1 i_2 .. i_{j-1} l2}(dummy_{i_1 i_2 .. i_{j-1} l}))$

// Use the dummy keys to send complementary keys
for$(x=1; x \le d_2; x=x+1)$
if $(x \neq l \wedge x \neq i_j)$
send $dummy_{i_1 i_2 .. i_{j-1} l}(c'_{i_1 i_2 .. i_{j-1} x})$

Changing the logical hierarchy keys at level h_1
send $c'_{i_1 i_2 .. i_{h_1} 1}(k'_{i_1 i_2 .. i_{h_1}})$
send $c'_{i_1 i_2 .. i_{h_1} 2}(k'_{i_1 i_2 .. i_{h_1}})$

Changing the logical hierarchy keys at other levels
for $(j=h_1-1; j>0; j=j-1)$
for $(l=1; l \le d_1; l=l+1)$ send $k'_{i_1 i_2 .. i_{l-2} i_j l}(k'_{i_1 i_2 .. i_j})$

[2]Encryption using a complementary key such as c_{113} is only possible only at level $h-1$. For levels j, $h_1 < j < h-1$, no such complementary key exists. Hence, we need to use a dummy key for sending complementary keys even in the subtree from where the user left. For simplifying the algorithm, we use the dummy keys at all levels.

Analysis of the above algorithm. Now, we discuss the algorithm in detail and identify the costs (in terms of encryptions and messages). At the h^{th} level, we use the individual user keys to send the required complementary keys. The leaving user had d_2-1 keys (of the h^{th} level) and these keys need to be changed. Each of these keys is sent to d_2-2 users and, hence, the number of encryptions required for this level is $(d_2-1)(d_2-2)$. In terms of messages, (d_2-1) messages are sufficient, one for each remaining user. Thus, we have

Observation 3.2 The number of encryptions required at the lowest level is $(d_2-1)(d_2-2)$. □

Observation 3.3 The number of messages required at the lowest level is (d_2-1). □

Now, we consider the tasks involved in changing the keys at level j, $h_1 < j < h$. At these levels, we need to change the complementary keys $c_{i_1 i_2 .. i_{j-1} l}$ where $l \neq i_j$ as these keys were with the leaving user. For these levels, we set up a dummy key before transmitting the new complementary keys. Depending upon the value of d_2, setting these keys reduces the number of encryptions required. To see this, consider the transmission of c'_{13} and c'_{14} to the descendents of c_{12} (cf. Figure 1). Without such a dummy key, we would need to encrypt c'_{13} and c'_{14} by both c_{121} and c_{122}. And, more generally, we would need to perform $2(d_2-2)$ encryptions. By setting the dummy key however, we need 2 encryptions to send the dummy key and (d_2-2) encryptions to send the complementary keys. It follows that if $d_2 > 4$, the number of encryptions are reduced if we set up the dummy key. And, if $d_2 = 4$, the number of encryptions remain the same.

To change the complementary keys at level j, $h_1 < j < h$, we observe that d_2-1 keys at level j are changed. All these keys need to be sent into the subtree $c_{i_1 i_2 .. i_{j-1} i_j}$ from where the user left. From the above discussion, the number of encryptions for this part is $(min(2+d_2-1, 2(d_2-1)))$. For the sibling trees rooted at $c_{i_1 i_2 .. i_{j-1} l}$, $l \neq i_j$, d_2-1 of these need to be sent. The number of encryptions for this part is $(d_2-1)(min(2+d_2-2, 2(d_2-2)))$. Thus, we have

Observation 3.4 The number of encryptions to change the complementary keys at levels j, $h_1 < j < h$, is $(\ min(2+d_2-1, 2(d_2-1)) \ + \ (d_2-1)(min(2+d_2-2, 2(d_2-2))) \ )(h_2-1)$. □

If $d_2 \geq 4$, the above equation can be simplified as follows.

Corollary 3.5 If $d_2 \geq 4$, the number of encryptions to change complementary keys at levels j, $h_1 < j < h$, is $(\ d_2+1 \ + \ (d_2-1)d_2 \ )(h_2-1)$. □

If we are interested in reducing the message transmissions only then the dummy keys should not be set up in the above scenario. For example in Figure 1, to transmit c'_{13} and c'_{14} to the descendents of c_{12}, we need two messages, $c_{121}(c'_{13}, c'_{14})$ and $c_{122}(c'_{13}, c'_{14})$. More generally, we need $2d_2$ messages at each level. With this modification, we have

Observation 3.6 The number of messages to change the complementary keys at levels j, $h_1 < j < h$, is $2d_2(h_2-1)$. □

After changing the complementary keys, we change the keys in the logical key hierarchy. At the h_1^{th} level, we use two of the complementary keys at level h_1+1. For the remaining levels, the algorithm is same as that in [3]. Thus, we have

Observation 3.7 The number of encryptions to change the logical keys is $d_1(h_1-1)+2$. □

Observation 3.8 The number of messages to change the logical keys is $d_1(h_1-1)+2$. □

Based on Observations 3.2, 3.5 and 3.7, we have

Observation 3.9 The number of encryptions for the non-critical path is $d_1(h_1-1)+2 \ + \ (d_2^2+1)(h_2-1) \ + \ (d_2-1)(d_2-2)$. □

And, based on Observations 3.3, 3.6 and 3.8 we have

Observation 3.10 The number of messages for the non-critical path is $2d_2h_2 - d_2 + d_1(h_1-1) + 1$. □

4 Second Algorithm: Complete Logical and Complementary Keys

In this section, we consider the question: "Had we maintained both types of keys at each node (respectively, certain nodes) in the tree in Section 3, could it have reduced the cost of the critical path and/or the non-critical path"? The motivation for increasing the complementary keys arises from the fact that these keys helped in reducing the cost of the critical path. The motivation for increasing the logical keys arises from the fact that these keys could simplify the transmission of the complementary keys.

Similar to the algorithm in Section 3, we first discuss the steady state configuration. Then, we discuss the join and leave respectively.

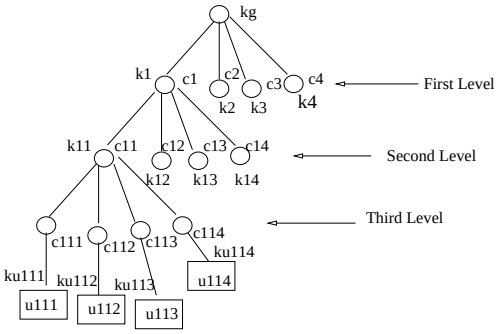

Figure 2. Representation of Users and Keys

Steady state configuration. Once again, the users and the keys are organized in a tree where users are at leaf nodes. Each internal node is associated with two keys; one for the logical key hierarchy and one for complementary key hierarchy. The user $u_{i_1 i_2 \ldots i_h}$ gets the following keys.

- Group key k_g
- Logical keys k_{i_1}, $k_{i_1 i_2}$, ..., $k_{i_1 i_2 .. i_{h-1}}$, and
- Complementary keys
  - c_l where $l \neq i_1$
  - $c_{i_1 l}$ where $l \neq i_2$
  - ...,
  - $c_{i_1 i_2 \ldots i_{h-1} l}$ where $l \neq i_h$

Join. The algorithm for join is similar that in Section 3 and, hence, is omitted.

Leave. First, we consider the example in Figure 2 to illustrate the process of rekeying when a user leaves.

Suppose that user u_{111}, has left the group. To preserve forward secrecy, the controller generates new values for all the keys held by u_{111}. The user u_{111} has $c_{112}, c_{113}, c_{114}, c_{12}, c_{13}, c_{14}, c_2, c_3, c_4, k_{11}, k_1$ and k_g. Let the corresponding primed keys be the new keys.

Critical Path. The distribution of group key k'_g is done entirely with the complementary keys. The controller sends the following rekey messages.

- $c_{111}(k'_g), c_{11}(k'_g), c_1(k'_g)$.

We generalize this scheme for a tree of height h. The algorithm is as follows.

```
Critical Path
    // For all the h levels
    for(j = h; j ≥ 1; j = j − 1)   send c_{i1 i2..ij}(k'_g)
```

The above algorithm ensures that all users except the leaving user can get the new group key. Thus, we have

Observation 4.1 The number of encryptions (respectively messages) for the critical path is h . □

Non-Critical Path. In Figure 2, at the lowest level, the complementary keys $c_{112}, c_{113}, c_{114}$ are changed. The controller encrypts the complementary keys each user needs with its individual key, i.e., it sends the following messages.

- $k_{u112}(c'_{113}, c'_{114}), k_{u113}(c'_{112}, c'_{114}), k_{u114}(c'_{112}, c'_{113})$

At the second level, the controller needs to send $c'_{12}, c'_{13}, c'_{14}$ and k'_{11}. The controller sends k'_{11} first by encrypting it with c_{111}. This ensures that all users except u_{111} can obtain this key. Then, it distributes the complementary keys using k'_{11}, k_{12}, k_{13} and k_{14}. The same approach is followed at the first level with the exception that k'_1 needs to be encrypted using c_{11} and c'_{12}. Thus, the controller sends:

- $c_{111}(k'_{11})$
- $k'_{11}(c'_{12}, c'_{13}, c'_{14}), k_{12}(c'_{13}, c'_{14}), k_{13}(c'_{12}, c'_{14}), k_{14}(c'_{12}, c'_{13})$
- $c_{11}(k'_1), c_{12}(k'_1)$
- $k'_1(c'_2, c'_3, c'_4), k_2(c'_3, c'_4), k_3(c'_2, c'_4), k_4(c'_2, c'_3)$

We generalize the above algorithm for a tree of height h. The algorithm starts at the bottom of the tree. At each level, we use the complementary keys of the lower level to distribute the logical key. To distribute the complementary keys at this level, we use the logical keys at the same level. The algorithm is as follows (We let d be the (maximum number of) children at any internal node and note that if a key has not changed the primed key is the same as the original key.)

```
    Changing keys at the lowest level
for(l = 1; l ≤ d; l = l + 1)
    for(k = 1; k ≤ d; k = k + 1)
        if(k ≠ l ∧ k ≠ i_h ∧ l ≠ i_h)
            send k_{u_{i1 i2 ... i_{h−1} l}}(c'_{i1 i2 ... k})

    Changing keys at all other levels
for(j = h − 1; j ≥ 1; j = j − 1)
    // send logical key
    send c'_{i1 i2 ... ij 1}(k'_{i1 i2 ... ij})
    send c'_{i1 i2 ... ij 2}(k'_{i1 i2 ... ij})

    // send complementary keys
    for(l = 1; l ≤ d; l = l + 1)
        for(x = 1; x ≤ d; x = x + 1)
            if(l ≠ x and l ≠ i_j) send k'_{i1 i2 ... ij−1 x}(c'_{i1 i2 ... ij−1 l})
```

Analysis of the above algorithm. As in Section 3, $(d-1)$ messages and $(d-1)(d-2)$ encryptions are required at the lowest level.

At other levels, we first send the keys from the logical key hierarchy. The logical key at level j is sent using the complementary keys at level $j+1$. At each level, it takes at most 2 messages for this task. Thus, the number of messages (respectively, encryptions) to change the logical keys is $2(h-1)$. After changing the logical keys at level j, we change the complementary keys at that level j by using the logical keys at level j. At each level, there are $d-1$ complementary keys that need to be changed. All these keys need to be sent into the subtree from where the user left. And, $d-2$ of these keys need to be sent in other trees. Thus, at each level, we need d messages and $(d-1)+(d-1)(d-2)$ encryptions. Combining all the encryption costs above, the cost of encryption is $((d-1)+(d-1)(d-2))(h-1) + 2(h-1) + (d-1)(d-2)$. Simplifying this, we get

Observation 4.2 The number of encryptions for the non-critical path is $(d-1)(d-2)h + (d+1)(h-1)$. □

Combining the messages in the steps above, the total messages is $d(h-1) + 2(h-1) + (d-1)$. Simplifying this, we get

Observation 4.3 The number of messages for the non-critical path is $dh+2h-3$. □

5 Comparing Critical and Total Costs

Now, we compare the algorithms in Sections 3 and 4 to that in [2, 3]. The total cost for our algorithms is the sum of the critical cost and the non-critical cost. For simplicity, in this comparison, we let $d_1 = d_2 = d$ and $h_1 = h_2 = h/2$.

For [2], computing the critical and non-critical cost is straightforward. However, since [3], was intended to minimize the total cost (and not the critical cost), there are two ways to compute the critical and total cost. The algorithm in [3] rekeys from the bottom of the tree and the group key is changed at the end. Computing thus, the cost of the critical path (and the cost of the total rekeying) is dh.

However, one can easily modify [3] so that the group key is changed first. To distribute the group key first, we need to encrypt the group key using $d-1$ keys at each level in the tree. Hence, computing thus, the critical cost will be $(d-1)h$. After the group key is changed, however, we need to perform $d(h-1)$ encryptions to change other keys. It follows that if we modify [3] to reduce the critical cost, the total cost will be almost double of the original cost. Hence, for the following table, we use the numbers for the unmodified algorithm of [3].

	LKH [3]	Complementary keys [2]	Algorithm in Section 3	Algorithm in Section 4
Critical path (messages)	dh	1	$dh/2$	h
Critical path (encryptions)	dh	1	$dh/2$	h
Total rekeying (messages)	dh	$n-1$	$2dh - 2d + 1$	$dh+3h-3$
Total rekeying (encryptions)	dh	$(n-1)*(n-2)$	$d^2h/2 + dh - 4d + h/2 + 3$	$d^2h-2dh+4h-d-1$

Now, we compare the costs for particular values of d and h. The cost comparison is as follows:

Group Size: $4^4 (n = 256)$	LKH [3]	Complementary keys [2]	Section 3	Section 4
Critical path (messages)	16	1	8	4
Critical path (encryptions)	16	1	8	4
Total rekeying (messages)	16	255	25	25
Total rekeying (encryptions)	16	$64k$	37	43

Group size: $4^6 (n = 4096)$				
Critical path (messages)	24	1	12	6
Critical path (encryptions)	24	1	12	6
Total rekeying (messages)	24	4095	41	39
Total rekeying (encryptions)	24	$16m$	62	67

Group Size: $5^4 (n = 625)$				
Critical path (messages)	20	1	10	4
Critical path (encryptions)	20	1	10	4
Total rekeying (messages)	20	624	31	29
Total rekeying (encryptions)	20	$388k$	55	70

From the above tables, the cost of the critical path in our algorithms is less than that in [3]. However, the total cost has increased. As an example, if we consider the case where $d = 4$ and $h = 4$ the algorithm in Section 3 reduces the critical cost from 16 to 8. However, at the same time the total cost increases from 16 messages to 25 messages. The algorithm in Section 4 reduces the critical cost further to 4 while leaving the number of messages to be 25 only. By comparison, if we want to further reduce the critical cost to 1, the total cost increases to 255 messages.

The number of encryptions in our algorithms also show the same pattern; the number of encryptions on the critical path are reduced; e.g., in a group of size 4^6, the critical cost can be reduced from 24 to 6 while increasing the total cost from 24 to 67. If we need to reduce the critical cost further, we need 16 million encryptions. Thus, our algorithms allow us to reduce the critical cost while keeping the total cost manageable.

6 Conclusion and Future Work

In this paper, we focused on the problem of reducing the critical cost of rekeying in a multicast group. Towards this end, we presented two algorithms that combined the logical key hierarchy with the complementary key hierarchy.

In both algorithms, we organized the keys in a tree. In the first algorithm (cf. Section 3), we divided the tree into two parts; in one part the logical key hierarchy was used while in another the complementary keys were used. In this algorithm, the cost of the critical path was half of that in [3]. The total encryption cost for this algorithm, however, increased by a factor of $\frac{d}{2}$, where d is the number of children a node has. And, the number of messages required was at most twice of that in [3].

Subsequently, in the second algorithm (cf. Section 4), we associated two keys with all internal nodes in the tree. In this case, the cost of the critical path was further reduced. In this algorithm, the cost of the critical path was reduced to $\frac{1}{d}^{th}$ of that in [3] where d is the degree of the key tree. Moreover, the number of total messages was only slightly increased from dh to $(d+3)h$ where h is the height of the tree. The cost in both these algorithms is significantly less than that in [1, 2] where $O(d^h)$ messages are required.

In other related work, in [4], authors have presented an algorithm where several (sub)groups are maintained and the communication across (sub)groups is handled by (sub)group controllers. Our work differs from that in [4] in that once the group keys are established, the group controller need not participate in group communication. By contrast in [4], the (sub)group controllers need to decrypt/encrypt messages. The algorithms in [5,6] are also based on key trees. However, they are restricted to binary trees. Also, unlike the algorithms in [7, 8] where the users outside the group can obtain messages sent to a group, in our solutions (as well as in [3–6]) ensure that only current members can obtain messages sent to the group.

Elsewhere [9], we also discussed extensions of these algorithms and showed how they can assist in heterogeneous systems where user requirements and capabilities —e.g., the need for lower critical cost, computing power, (in)ability to maintain several keys, and the time for which they are likely to continue to be in the group— vary. We also expect that our algorithms will work well with proxy frameworks such as [10] where certain users interact with the rest of the group through a proxy. By grouping the users appropriately and choosing appropriate algorithms for different subgroups, such a framework will permit adaptive implementation based on the user requirements.

References

[1] H.Harney and C.Muckenhirn. Group key management protocol (GKMP) specifi cation. RFC 2093, July 1997.

[2] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key management for multicast: Issues and architectures. RFC 2627.

[3] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 2000.

[4] S.Mittra. Iolus: A framework for scalable secure multicasting. In *Proc. ACM SIGCOMM'97*, pages 277–288, 1997.

[5] Marcel Waldvogel, Germano Caronni, Dan Sun, Nathalie Weiler, and Bernhard Plattner. The versakey framework: Versatile group key management. *IEEE Journal on Selected Areas in Communications*, 1999.

[6] D.McGrew and A.Sherman. Key establishment in large dynamic groups using one-way function trees. Manuscript.

[7] Michel Abdalla, Yuval Shavitt, and Avishai Wool. Key management for restricted multicast using broadcast encryption. *IEEE/ACM Transactions on Networking*, 8(4), August 2000.

[8] S. Setia and Sushil Jajodia Samir Koushish. Kronos: A scalable group re-keying approach for secure multicast. In *IEEE Symposium on Security and Privacy*, pages 215–228, 2000.

[9] Sandeep S. Kulkarni and Bezawada Bruhadeshwar. Critical path in secure multicast for dynamic groups. Technical Report MSU-CSE-02-9, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, March 2002.

[10] P. K. McKinley and A. P. Mani. A study of proxy-based adaptive forward error correction for collaborative computing on wireless lans. *IEEE Symposium on Applications and the Internet (SAINT), San Diego, California*, January 2001.