

**Adaptive Rekeying for Secure Multicast\***Sandeep KULKARNI<sup>†</sup> and Bezawada BRUHADSHWAR<sup>†</sup>, *Nonmembers*

**SUMMARY** 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 group 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 a family of algorithms that show the tradeoff between the cost of the critical path and the cost of the non-critical path. Our solutions allow the group controller to choose the appropriate algorithm for key distribution by considering the requirements on critical and non-critical cost. In our solutions, the group controller can dynamically change the algorithm for key distribution to adapt to changing application requirements. Moreover, we argue that our solutions allow the group controller to effectively manage heterogeneous groups where users have different requirements/capabilities.

**key words:** *Group Communication, Secure Multicast, Critical Path, Adaptive Rekeying*

**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 the data is shared with several authorized users, are group-oriented. Hence, encryption of the data is necessary to secure it from unauthorized access.

One way to prevent unauthorized access by intruders is to use the simple group key management protocol [2], GKMP. In GKMP, data is encrypted using a group key that is known only to the participating users. Towards this end, a controller of the group generates

and distributes the group key —using the personal key of each user— to all the users in the group. And, since the group key is known only to the participating users, any intruder eavesdropping on the communication cannot decrypt it.

If users join and leave the group, the group controller needs to modify the group key to reflect the current membership of the group. For example, when a user joins (respectively, leaves) the group, the group key needs to be changed so that the joining (respectively, leaving) user cannot access the past (respectively, future) communication.

The tasks involved when a user joins are straightforward in GKMP; the group controller sends the new group key to (1) the joining user through a secure channel (e.g., by encrypting it with the personal key of the joining user) and (2) to the users in the old group by encrypting it with the old group key. However, when a user leaves, the cost of rekeying is high; the group controller sends the new group key to each user separately using a secure channel. Thus, GKMP lacks scalability when groups are large and dynamic.

One approach to reduce the cost of the leave operation is to use the complementary variables technique [4]. In this technique, the group controller assigns a unique identifying number,  $u_1, u_2, \dots, u_n$ , to each of the users. It also generates the keys  $c_1, c_2, \dots, c_n$ . 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 cost of changing the group key is low in this technique, extra overhead is incurred while changing the keys  $c_1, c_2, \dots, c_{i-1}, c_{i+1}, \dots, c_n$ . 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: critical cost, i.e., the cost to change the group key, and non-critical cost, i.e., 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, changing the group key, is on the critical path to restoring the group communication after a user has left. By contrast, the second part, changing the other keys, is on the non-critical path; the process of rekeying on the non-critical path can be done in *background*, i.e., after the group communication is restored.

In this paper, we focus on a family of algorithms for

Manuscript received March 20, 2003.

Final manuscript received May 30, 2003.

<sup>†</sup>The authors are members of the Software Engineering and Network Systems (SENS) Laboratory at the Department of Computer Science and Engineering in Michigan State University.

\*Email: {sandeep, bezawada}@cse.msu.edu

Web: <http://www.cse.msu.edu/~{sandeep, bezawada}>

Tel: +1-517-355-2387, Fax: +1-517-432-1061

A preliminary version of this paper appears in [1].

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.

reducing the critical cost while keeping the non-critical cost manageable. Thus, based on the needs of the application, the group controller can select an appropriate algorithm for key distribution so that the critical cost and the non-critical cost are within acceptable limits. Moreover, our solutions are adaptive, i.e., if the application requirements change, it is possible to change the algorithm for key distribution accordingly.

Our approach is based on the use of logical keys [3] and complementary keys [4]. The critical cost in the complementary key technique is  $O(1)$ . However, the total (critical + non-critical) cost in this technique involves sending  $O(n)$  keys to  $O(n)$  users and requires  $O(n^2)$  encryptions. By contrast, in [3], the keys are arranged in a tree and the critical (respectively, total) cost of rekeying is  $O(dh)$  where  $d$  is the degree of the tree and  $h$  is the height of the tree. We propose a new approach that incorporates both these approaches to reduce the critical cost while keeping the non-critical (respectively, total) cost manageable.

**Contributions of the paper.** The main contributions of this paper are as follows:

- We present a family of algorithms that are based on different ways to organize the logical keys and complementary keys in a key tree. Then, we identify the effect of these algorithms on the critical and the non-critical cost during group reorganization. Depending upon the application requirements about critical and non-critical cost, the group controller can choose an appropriate algorithm from this family. Moreover, if collusion between leaving users is expected to be unlikely, we show that the total cost of rekeying can be reduced further so that it is even lower than that in [3].
- As an illustration, we present four algorithms from this family and compute the cost of rekeying. These algorithms illustrate the tradeoff between the critical cost and the non-critical cost.
- We also show how the group controller can adapt to changing application requirements to alter the algorithm for key distribution. Our approach also allows the group controller to provide differential service, where users who are likely to be *permanent* are less affected during changes in group membership than users who are likely to be *transient*. We also argue that our algorithms can be used in heterogeneous groups where the users have different capabilities and different requirements.

**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. In Section 3, we present a family of algorithms that identify the tradeoff between the critical cost and the non-critical cost. Then, in Section 4, we present four sample algorithms from this family and compare them with the solutions in [3,4]. In Section 5, we discuss tech-

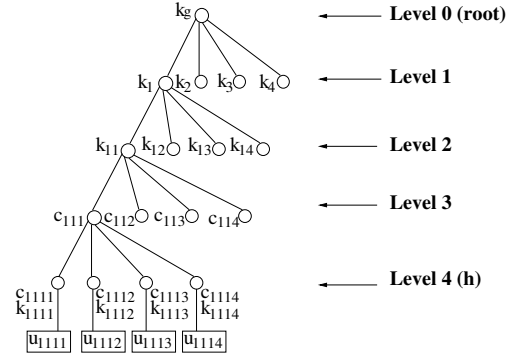


Fig. 1 Representation of users and keys

niques that reduce the non-critical cost further. Finally, in Section 6, we discuss the adaptivity provided by our solutions, and make concluding remarks in Section 7.

## 2. Notations

Similar to the logical key hierarchy algorithm [3], we arrange the users and the keys in a tree (cf. Figure 1). We use  $d$  to denote the number of children of a node in the tree and  $h$  to denote the height of the tree. The leaf nodes in this tree correspond to the users. Each tree node may be associated with a key related to one or both hierarchies. We use  $k_x$  to denote the key associated with the logical key hierarchy at node  $x$ . And, we use  $c_x$  to denote the key associated with the complementary key hierarchy at node  $x$ .

For the levels where we use the logical key hierarchy, the user obtains the keys on the path to the root. For the levels where we use the complementary key hierarchy, the user gets the keys associated with the siblings of its ancestors; it does not get the keys associated with its ancestor. Note that at level 0 (root), only logical keys can be used; the logical key at level 0 is the group key. Also, at the lowest level ( $h^{th}$  level), logical keys must be used (alone or with complementary keys). The key at this level is known only to the corresponding user and the group controller. Finally, we use *dummykey* to denote a temporary key.

As an illustration, in Figure 1, we have shown a partial tree of height 4. The logical keys are used at levels 0, 1, 2 and 4, and the complementary keys are used at levels 3 and 4. In this system, user  $u_{1112}$  gets the group key ( $k_g$ ), the keys associated with the logical key hierarchy ( $k_1, k_{11}$  and  $k_{1112}$ ), and the keys associated with the complementary key hierarchy ( $c_{112}, c_{113}, c_{114}, c_{1111}, c_{1113}$  and  $c_{1114}$ ).

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. The encryption of messages  $m_1, m_2, \dots$  by key  $k_i$  is denoted by  $k_i(m_1, m_2, \dots)$ . We compute the complexity of our algorithms in terms of the number of encryptions and the number of messages. The number

of messages are used to identify the number of “units” that need to be routed, if the network is not broadcast in nature. Since a message encrypted with key  $k$  is intended for users that share  $k$ , in the computation 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  $k_1, k_2, \dots, k_m$  be transmitted by encrypting them with key  $k$ , only one message,  $k(k_1, k_2, \dots, k_m)$ , is sent. However, for this message, the number of encryptions is  $m$ .

### 3. Arrangements of Logical and Complementary Keys

In this section, we present a family of algorithms for secure multicast in dynamic groups. Towards this end, we present different ways of arranging logical and complementary keys in a key tree (cf. Figure 1). Each arrangement results in an algorithm in this family. Hence, for each arrangement, we identify how these keys are used when a user leaves the group, and identify the corresponding critical and non-critical cost. When a user leaves, the group controller first changes the group key. This is achieved by using the keys associated with the ancestor (respectively, the siblings of the ancestor) of the leaving user. After changing the group key, the group controller changes the other keys that the leaving user had. Towards this end, the group controller begins from the lowest level in the key tree. Then, the new keys at the lowest level are used to change the keys at the next level, and so on. Since this process is bottom-up, the cost of changing keys at a level depends on the keys used at the next lower level.

Based on the above discussion, while changing the group key, the cost incurred at a given level depends only on the keys at that level. While changing the remaining keys, we separately consider the cost of changing keys at the lowest level and the cost of changing keys at other levels.

We proceed as follows. First, in Section 3.1, we compute the critical cost based on the keys used at a given level. Then, in Section 3.2, we compute the non-critical cost for changing the keys at the lowest level. Finally, in Section 3.3, we compute the non-critical cost for changing the keys at other levels. For this discussion, without loss of generality, we assume that the leaving user is the leftmost user in the key tree (cf. Figure 2). Also, we use  $k_x, c_x$  (unprimed) to denote the old keys and use  $k'_x, c'_x$  (primed) to denote the corresponding new keys. If a key  $k_x$  is not changed during rekeying,  $k'_x$  is the same as  $k_x$ .

#### 3.1 Computing Critical Cost

In this section, we compute the critical cost in changing the group keys using the keys at a given level, say

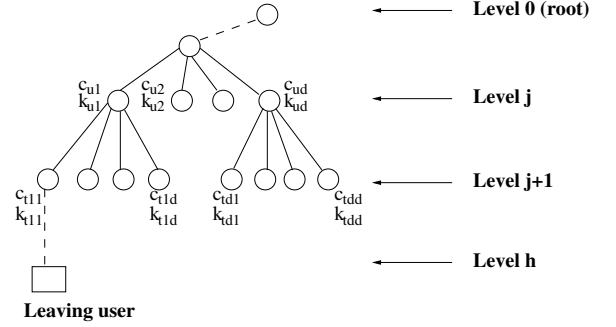


Fig. 2 Representation of key levels

$u$ . Let  $k_{u1}, k_{u2}, \dots, k_{ud}$  be the logical keys (if used) at this level. Likewise, let  $c_{u1}, c_{u2}, \dots, c_{ud}$  be the complementary keys (if used) at this level. Next, we consider three cases based on the types of keys used at level  $u$ .

**Case 1.1: Logical keys at level  $u$ .** To distribute the new group key,  $k'_g$ , using the keys at level  $u$ , the group controller selects those keys at level  $u$  that are not shared with the leaving user. Then, it encrypts the new group key with these keys. Thus, the group controller sends the following messages:

$$k_{u2}(k'_g), k_{u3}(k'_g), \dots, k_{ud}(k'_g)$$

Based on the above calculation, the critical cost for level  $u$  is  $(d-1)$  encryptions. And, the number of messages sent is  $(d-1)$ .

**Case 1.2: Complementary keys at level  $u$ .** Once again, in this case, the group controller selects those keys at level  $u$  that the leaving user does not have. For the case where complementary keys are used at level  $u$ ,  $c_{u1}$  suffices for this purpose. Thus, the group controller sends the following message:

$$c_{u1}(k'_g)$$

Thus, the critical cost for level  $u$  is one encryption. The number of messages sent is one. Thus, the critical cost is low when complementary keys are used.

**Case 1.3: Logical and complementary keys at level  $u$ .** In this case, the solution is identical to that in Case 1.2.

**Theorem 3.1** For each level, if the group controller distributes the new group key using the appropriate case as given above then all the remaining users will get the new group key. And, the leaving user will not get the new group key.  $\square$

#### 3.2 Computing Non-critical Cost at the Lowest Level

Once again, let  $k_{h1}, k_{h2}, \dots, k_{hd}$  be the logical keys at the lowest level. Likewise, let  $c_{h1}, c_{h2}, \dots, c_{hd}$  be the complementary keys (if used) at this level. As mentioned in Section 2, we must use logical keys at this

level; the logical key at this level is the personal key of the user and it is shared only between the user and the group controller. Hence, we consider the two cases based on the types of keys used at the lowest level.

**Case 2.1: Logical keys at level  $h$ .** In this case, no keys need to be changed as the leaving user only has the key  $k_{h1}$ . And, no other user in the system has this key. Thus, the non-critical cost for this case is zero.

**Case 2.2: Logical and complementary keys at level  $h$ .** In this case, the complementary keys at level  $h$  need to be changed. For the user located at  $c_{h2}$ , this can be achieved by sending the following message (Similar messages are sent for the users located at  $c_{h3}, \dots, c_{hd}$ ):

$$k_{h2}(c'_{h3}, c'_{h4}, \dots, c'_{hd})$$

Thus, the non-critical cost for this case is  $(d-1)(d-2)$  encryptions. And, the number of messages sent is  $(d-1)$ .

### 3.3 Computing Non-critical Cost at Other Levels

As mentioned above, the cost of changing the keys at a level (other than the lowest level) depends on the keys used at that level and the keys used at the next lower level. Therefore, in this subsection, we consider different cases based on the keys arranged at two consecutive levels, say  $j$  and  $j+1$ . For the following discussion, we call these the upper level ( $u$ ) and the lower level ( $l$ ) respectively.

We use  $k_{u1}, k_{u2}, \dots, k_{ud}$  to denote the logical keys at the upper level and, likewise,  $k_{l11}, k_{l12}, \dots, k_{l1d}$  to denote the logical keys at the lower level. Similarly, we use  $c_{u1}, c_{u2}, \dots, c_{ud}$  to denote the complementary keys at the upper level and  $c_{l11}, c_{l12}, \dots, c_{l1d}$  to denote the complementary keys at the lower level. Note that keys  $k_{l1*}$  are in the tree rooted at  $k_{u1}$ , keys  $k_{l2*}$  are in the tree rooted at  $k_{u2}$ , and so on.

We proceed as follows. In Section 3.3.1, we consider the case where logical keys are used at the upper level. Then, in Section 3.3.2, we consider the case where complementary keys are used at the upper level. Finally, in Section 3.3.3, we consider the case where both keys are used at the upper level.

#### 3.3.1 Logical Keys at the Upper Level

We consider three cases based on the keys used at the lower level.

**Case 3.1.1: Logical keys at the lower level.** The key  $k_{u1}$  needs to be changed at the upper level. The group controller can achieve this by sending the following messages (Note that since we are using the new keys at the lower level, by induction, the leaving user does not have them.):

$$k'_{l11}(k'_{u1}), k'_{l12}(k'_{u1}), \dots, k'_{l1d}(k'_{u1})$$

Based on the above calculation, the non-critical cost is  $d$  encryptions. The number of messages sent is  $d$ . The number of users rooted at  $k_{u1}$  is  $d^{h-j}$ . Each of these users decrypts one of the above messages to obtain  $k'_{u1}$ . Thus, the total number of decryptions by the users in this case is  $d^{h-j}$ .

**Special subcase at level  $(h-1)$ .** If the group controller is changing the logical key at level  $(h-1)$ , then the number of users that need the new logical key at level  $(h-1)$  is  $(d-1)$ . Thus, it would be possible to save one encryption and one message while changing keys at level  $(h-1)$ . A similar subcase also applies to Cases 3.2.1 and 3.3.1.

**Case 3.1.2: Complementary keys at the lower level.** In this case, we can use two complementary keys to send a message to all the users rooted at  $k_{u1}$ . Observe that each user in the subtree rooted at  $k_{u1}$  has at least one of these keys. Thus, the group controller sends the following messages:

$$c'_{l11}(k'_{u1}), c'_{l12}(k'_{u1})$$

Thus, the non-critical cost in this case is two encryptions. The number of messages sent is two. The number of decryptions by the users is  $d^{h-j}$ .

**Special subcase at level  $(h-1)$ .** If the group controller is changing the logical key at level  $(h-1)$  and if complementary keys are used at level  $h$  then we can use  $c_{l11}$  to send the new key to the users rooted at  $k_{u1}$ . Thus, it would be possible to save one encryption and one message while changing keys at level  $(h-1)$ . A similar subcase also applies to Cases 3.1.3, 3.2.2, and 3.3.2. And, in Case 3.2.2, the reduction is two encryptions and two messages.

**Case 3.1.3: Complementary and logical keys at the lower level.** This case is identical to Case 3.1.2. In fact, we observe that the logical keys at the lower level do not affect the non-critical cost. Hence, in Sections 3.3.2 and 3.3.3, we omit the discussion for the case where both types of keys are used at the lower level.

#### 3.3.2 Complementary Keys at the Upper Level

Once again, we consider two cases based on the keys used at the lower level.

**Case 3.2.1: Logical keys at the lower level.** The keys known to the leaving user at the upper level are,  $c_{u2}, c_{u3}, \dots, c_{ud}$  and, hence, the group controller needs to change them. To distribute the new keys to the users rooted at  $c_{u1}$ , the lower level keys,  $k'_{l11}, k'_{l12}, \dots, k'_{l1d}$ , are selected. The group controller needs to encrypt the new keys,  $c'_{u2}, c'_{u3}, \dots, c'_{ud}$ , with each of these keys and send them to these users. However, the direct approach for sending these keys is expensive in terms of the number of encryptions performed by the group controller. Hence, we propose a technique that reduces the encryption cost for the group controller. In this technique, the group controller, first, sends a dummy key

encrypted with each of the logical keys at the lower level. Since this dummy key is now known to all the users (except the leaving user) rooted at  $c_{u1}$ , the group controller uses this key to distribute the new complementary keys. Thus, the group controller sends the following messages:

$$k'_{l11}(dummykey_1), \dots, k'_{l1d}(dummykey_1) \\ dummykey_1(c'_{u2}, c'_{u3}, \dots, c'_{ud})$$

The new keys also need to be sent out to users rooted at  $c'_{u2}, c'_{u3}, \dots, c'_{ud}$ . For example, the users rooted at  $c'_{u2}$  need to get  $c'_{u3}, c'_{u4}, \dots, c'_{ud}$ . The group controller achieves this by distributing a dummy key to the users and then using this dummy key to distribute the new keys. The group controller sends the following messages to users rooted at  $c'_{u2}$  (Similar messages are sent for users rooted at  $c'_{u3}, \dots, c'_{ud}$ ):

$$k_{l21}(dummykey_2), \dots, k_{l2d}(dummykey_2) \\ dummykey_2(c'_{u3}, c'_{u4}, \dots, c'_{ud})$$

Based on the above calculation, the non-critical cost is  $d + (d-1)(2d-1)$  encryptions. The number of messages sent is  $d(d+1)$ . The number of users rooted at  $c_{u1}$  is  $d^{h-j}$ . Each of these users decrypts one message to obtain the dummy key and  $(d-1)$  decryptions to obtain the new keys. Thus, the number of decryptions by the users rooted at  $c_{u1}$  is  $d^{h-j}(d)$ . Also, the users rooted at  $c_{u2}, c_{u3}, \dots, c_{ud}$  decrypt one message each to obtain the dummy key and  $(d-2)$  decryptions to obtain the new keys. The number of decryptions by these users is  $d^{h-j}(d-1)(d-1)$ . Thus, the total number of decryptions by the users in this case is  $d^{h-j}(d^2 - d + 1)$ .

**Case 3.2.2: Complementary keys at the lower level.** As in Case 3.1.2, two complementary keys at the lower level are sufficient for distributing the new keys. As before, the group controller initially distributes a dummy key using two complementary keys from the lower level. Then, it uses the dummy key to distribute the new keys at the upper level. Thus, for the users rooted at  $c'_{u1}$  the group controller sends the following messages (Similar messages are sent for users rooted at  $c'_{u2}, c'_{u3}, \dots, c'_{ud}$ ):

$$c'_{l11}(dummykey_3), c'_{l12}(dummykey_3) \\ dummykey_3(c'_{u2}, c'_{u3}, \dots, c'_{ud})$$

Thus, the non-critical cost is  $(d+1) + (d-1)d$  encryptions. The number of messages sent is  $3d$ . The number of decryptions by users is  $d^{h-j}(d^2 - d + 1)$ .

### 3.3.3 Complementary and Logical Keys at the Upper Level

**Case 3.3.1: Logical keys at the lower level.** The group controller needs to change  $c_{u2}, c_{u3}, \dots, c_{ud}$  and  $k_{u1}$ . First, the group controller sends  $k'_{u1}$  using keys at the lower level. Then, it uses  $k'_{u1}$  to send  $c'_{u2}, c'_{u3}, \dots, c'_{ud}$ . Thus, the group controller sends the following messages for users rooted at  $c_{u1}$ :

$$k'_{l11}(k'_{u1}), k'_{l12}(k'_{u1}), \dots, k'_{l1d}(k'_{u1}) \\ k'_{u1}(c'_{u2}, c'_{u3}, \dots, c'_{ud})$$

The group controller also needs to distribute some of the new keys to the users rooted at  $c'_{u2}, c'_{u3}, \dots, c'_{ud}$ . The group controller uses the keys,  $k_{u2}, k_{u3}, \dots, k_{ud}$ , to send the new keys to these users. Thus, the group controller sends the following message for users rooted at  $k_{u2}$  (Similar messages are also sent to users rooted at  $k_{u3}, k_{u4}, \dots, k_{ud}$ ):

$$k_{u2}(c'_{u3}, c'_{u4}, \dots, c'_{ud})$$

Based on the above calculation, the non-critical cost is  $(2d-1) + (d-1)(d-2)$  encryptions. The number of messages sent is  $2d$ . The number of decryptions by users is  $d^{h-j}(d^2 - 2d + 2)$ .

**Case 3.3.2: Complementary keys lower level.** To distribute  $k'_{u1}$ , the group controller uses two complementary keys from the lower level and sends the following messages:

$$c'_{l11}(k'_{u1}), c'_{l12}(k'_{u1})$$

The group controller then uses  $k'_{u1}$  to distribute the new complementary keys,  $c'_{u2}, c'_{u3}, \dots, c'_{ud}$ , by sending the following message:

$$k'_{u1}(c'_{u2}, c'_{u3}, \dots, c'_{ud})$$

For users rooted at  $c'_{u2}, c'_{u3}, \dots, c'_{ud}$ , the group controller uses the keys  $k_{u2}, k_{u3}, \dots, k_{ud}$  to distribute the new keys. For example, for users rooted at  $c'_{u2}$ , the group controller sends the following message (Similar messages are sent to users rooted at  $c'_{u3}, c'_{u4}, \dots, c'_{ud}$ ):

$$k_{u2}(c'_{u3}, c'_{u4}, \dots, c'_{ud})$$

Thus, the non-critical cost is  $(d+1) + (d-1)(d-2)$  encryptions. The number of messages sent is  $(d+2)$ . The number of decryptions by users is  $d^{h-j}(d^2 - 2d + 2)$ .

**Observations.** Based on the costs computed in this section, we observe the following:

1. If complementary keys are used at a level then the critical cost is reduced for that level.
2. If complementary keys are used at level  $j+1$  then adding logical keys at that level does not affect the cost of changing keys at level  $j$ .
3. If complementary keys are used at level  $j$  then adding logical keys at level  $j$  reduces the cost of changing keys at level  $j$ .

We use these observations to enable the group controller to dynamically change the algorithm for key distribution.

**Theorem 3.2** For each level, if the group controller distributes the new keys using the appropriate case given above then all the remaining users get the required new keys. And, the leaving user will not get any of the new keys.  $\square$

#### 4. Key Management Algorithms

Based on the keys used at different levels of the key tree, we get a different algorithm for dealing with security in a dynamic group. In this section, we present four of these algorithms.

We proceed as follows. In Section 4.1, we present the first algorithm where the user has logical keys for the upper  $\frac{h}{2}$  levels and complementary keys for the lower  $\frac{h}{2}$  levels. In Section 4.2, we present the second algorithm where the user has complementary keys for the upper  $\frac{h}{2}$  levels and logical keys for the lower  $\frac{h}{2}$  levels. In Section 4.3, we present the third algorithm where the user has complementary keys for every level in the key tree. Finally, in Section 4.4, we present the fourth algorithm where the user has both logical and complementary keys for every level in the key tree.

##### 4.1 Logical Keys Upper Half

In this algorithm, for each user, the group controller maintains logical keys at levels  $1, \dots, \frac{h}{2}$  and complementary keys at levels  $\frac{h}{2}+1, \frac{h}{2}+2, \dots, h$  of the key tree. Note that, as stated in Section 2, at the  $h^{th}$  level, now there are complementary keys as well as logical keys.

*Cost of critical path.* Observe that Case 1.1 applies for distributing the new group key for levels,  $1, \dots, \frac{h}{2}$  and Case 1.2 (or Case 1.3) applies for levels  $\frac{h}{2}+1, \frac{h}{2}+2, \dots, h$ . Thus, the critical cost is  $\frac{dh}{2}$  encryptions. And, the number of messages sent is  $\frac{dh}{2}$ .

*Cost of non-critical path.* From Case 2.2, the non-critical cost for the lowest level is  $(d-1)(d-2)$  encryptions and  $(d-1)$  messages. Observe that Case 3.2.2 applies (for changing keys) at levels  $(h-1)$  to  $(\frac{h}{2}+1)$ . Also, as discussed in Case 3.1.2, it is possible to reduce the encryptions and messages by two while changing keys at level  $(h-1)$ . Thus, the non-critical cost for these levels is  $(d^2-1) + (d^2+1)(\frac{h}{2}-2)$  encryptions and  $(3d-2) + 3d(\frac{h}{2}-2)$  messages. Case 3.1.2 applies at level  $\frac{h}{2}$ . The non-critical cost for this level is two encryptions and two messages. Case 3.1.1 applies for levels  $(\frac{h}{2}-1)$  to 1. The non-critical cost for these levels is  $d(\frac{h}{2}-1)$  encryptions and  $d(\frac{h}{2}-1)$  messages. Combining the rekeying costs for individual levels, the non-critical cost of this algorithm is  $\frac{h}{2}(d^2+d+1)-4d+1$  encryptions. And, the number of messages sent is  $(2dh-3d-1)$ .

##### 4.2 Complementary Keys Upper Half

In this algorithm, for each user, the group controller maintains complementary keys at levels  $1, 2, \dots, \frac{h}{2}$  and logical keys at levels  $\frac{h}{2}+1, \frac{h}{2}+2, \dots, h$  of the key tree.

*Cost of critical path.* Case 1.2 applies for levels  $1, 2, \dots, \frac{h}{2}$  and Case 1.1 applies for levels  $\frac{h}{2}+1, \frac{h}{2}+2, \dots, h$ . Thus, the critical cost is  $\frac{dh}{2}$  encryptions. And,

the number of messages sent is  $\frac{dh}{2}$ .

*Cost of non-critical path.* As we discussed in Section 3.2, the group controller need not change the logical keys at the lowest level. Case 3.1.1 applies for levels  $(h-1)$  to  $(\frac{h}{2}+1)$ . Also, as we discussed in Case 3.1.1, it is possible to reduce the number of encryptions and messages by one while changing the keys at level  $(h-1)$ . Thus, the non-critical cost for these levels is  $(d-1)+d(\frac{h}{2}-2)$  encryptions and  $(d-1)+d(\frac{h}{2}-2)$  messages. Case 3.2.1 applies for level  $\frac{h}{2}$ . The non-critical cost for this level is  $(2d^2-2d+1)$  encryptions and  $d(d+1)$  messages. Case 3.2.2 applies for levels  $(\frac{h}{2}-1)$  to 1. The non-critical costs for these levels is  $(d^2+1)(\frac{h}{2}-1)$  encryptions and  $3d(\frac{h}{2}-1)$  messages. Combining the rekeying costs for individual levels, the non-critical cost of this algorithm is  $\frac{h}{2}(d^2+d+1)+d^2-3d-1$  encryptions. And, the number of messages sent is  $(d^2+2dh-3d-1)$ .

##### 4.3 Complete Complementary Keys

In this algorithm, the user has complementary keys for every level (except level 0) in the key tree.

*Cost of critical path.* Case 1.2 (or Case 1.3) applies for all the levels. Thus, the critical cost is  $h$  encryptions. And, the number of messages sent is  $h$ .

*Cost of non-critical path.* Case 2.2 applies for the lowest level. Thus, the non-critical cost for the lowest level is  $(d-1)(d-2)$  encryptions and  $(d-1)$  messages. Case 3.2.2 applies for levels  $(h-1)$  to 1. Also, as we discussed in Case 3.1.2, it is possible to reduce the encryptions and messages by two while changing the keys at level  $(h-1)$ . Thus, the non-critical cost for these levels is  $(d-1)+(d-1)d+(d^2+1)(h-2)$  encryptions and  $(3d-2) + 3d(h-2)$  messages. Combining the rekeying costs for the individual levels, the non-critical cost of this algorithm is  $h(d^2+1)-3d-1$  encryptions. And, the number of messages sent is  $(3dh-2d-3)$ .

##### 4.4 Complete Logical and Complementary Keys

*Cost of critical path.* Case 1.3 applies for all the levels in the key tree. Thus, the critical cost is  $h$  encryptions. And, the number of messages sent is  $h$ .

*Cost of non-critical path.* Case 2.2 applies for the lowest level. Thus, the non-critical cost for the lowest level is  $(d-1)(d-2)$  encryptions and  $(d-1)$  messages. Case 3.3.2 applies for levels  $(h-1)$  to 1. Also, as we discussed in Case 3.1.1, it is possible to reduce the encryptions and the number of messages by one while changing the keys at level  $(h-1)$ . Thus, the non-critical cost for these levels is  $d+(d-1)(d-2)+((d+1)+(d-1)(d-2))(h-2)$  encryptions and  $d+1+(d+2)(h-2)$  messages. Combining the rekeying costs for the individual levels, the non-critical cost of this algorithm is  $h(d^2-2d+3)-d-2$  encryptions. And, the number of messages sent is  $(dh+2h-4)$ .

**Summary.** Now, we summarize the results from this section, in Figure 3, and compare these results to

	[3]	[4]	§ 4.1	§ 4.2	§ 4.3	§ 4.4
Critical (messages)	23	1	12	12	6	6
Critical (encryptions)	23	1	12	12	6	6
Total (messages)	23	4096	47	63	67	38
Total (encryptions)	23	$>10^7$	60	78	95	66

**Fig. 3** Complexity of the algorithms for  $4^6$  ( $d=4, h=6$ ) users

those in [3, 4]. In [3], the group key is distributed last and, hence, the critical cost is the same as the total cost. We show the performance of our algorithms on a sample group of size  $4^6$ . From the results for the sample group, we observe that: (1) The critical cost in our algorithms is lower than the algorithm in [3] and the total cost of rekeying is reasonable. For this group, we see that the critical cost in [3] is 23 encryptions whereas it is 6 encryptions in the algorithm of Section 4.4. (2) Although one can achieve a critical cost of 1 encryption by using the algorithm in [4], the non-critical cost in [4] increases by orders of magnitude when compared with our algorithms. (3) The algorithms in Section 4.3 and Section 4.4 have the lowest critical cost among all algorithms.

## 5. Reducing Non-critical Cost Further

In the algorithms presented in Section 4, the group controller distributes the new group key before distributing any other keys. In this section, we show that this fact can be used to reduce the non-critical cost further. We first describe two approaches to reduce the non-critical cost and then discuss the tradeoff involved in them. Our approaches are variations of those proposed in [5, 6].

*First approach.* Let  $k$  denote one of the keys that the leaving user,  $u$ , shared with other users. When  $u$  leaves the group, the group controller can change this key by sending the following message (where  $k'_g$  is the new group key and  $k'$  is the new version of key  $k$ ):

$$k'_g(k(k'))$$

Though this message can be received by every user in the group, only those users who have the old key  $k$  and the new group key  $k'_g$  can obtain  $k'$ . As we can see, the cost of this approach is less; only two encryptions (respectively, one message) for every key that the leaving user had. For example, using this approach the non-critical cost of the algorithm in Section 4.1, for a group size of  $4^6$ , is reduced from 48 to 24 encryptions and the number of messages sent is reduced from 35 to 12.

*Second approach.* The users can also generate the new key by using a one-way hash function, say  $f$ , that is known to all users. Thus, the users can generate the

new key by using the following function:

$$k' = f(k, k'_g)$$

If the group controller uses this approach then the non-critical cost (in terms of encryptions/messages) is zero as far as the group controller is concerned. If the number of users is  $4^6$  then the use of this approach in the algorithm in Section 4.4 will reduce the total encryptions/messages to 6. By contrast, 23 encryptions/messages are required for the solution in [3]. Moreover, even though the algorithm in [4] can be used to reduce the cost to 1, the number of keys stored by each user to achieve this cost is  $4^6$ . By contrast, our algorithm in Section 4.4 reduces the critical cost to 6 encryptions/messages while keeping the number of keys stored by users to 25. Note that the reduction achieved using this approach is not compatible with [3]; in [3], the new group key is distributed after all the other keys have been distributed.

**Tradeoff.** Compared to the algorithms in Section 4, the above approaches reduce the total cost of rekeying. However, collusion poses a difficulty in these approaches. Consider users  $u_m$  and  $u_n$  who are currently part of the group. Now, consider the case when  $u_m$  is removed from the group. In this case, the group controller changes all the keys that  $u_m$  is aware of.

Now, if  $u_m$  obtains the new group key  $k'_g$  from  $u_n$  then  $u_m$  also gets the new versions of the keys it had. With these keys, essentially,  $u_m$  continues to be a part of the group. Further, consider the case when  $u_n$  is removed from the group, at a later time. The group controller changes the group key and distributes the new group key,  $k''_g$ , to all the users in the group. Since  $u_m$  is effectively still in the group,  $u_m$  will get  $k''_g$ . If  $u_n$  gets  $k''_g$  from  $u_m$  then  $u_n$  will also get the new versions of the keys it had. Thus, it would be impossible to remove  $u_m$  and  $u_n$ . By contrast, in the algorithms presented in Section 4, knowledge of the new group key by  $u_m$  or  $u_n$  does not enable them to learn about the other keys.

It follows that if collusion is unlikely in a given system, then our algorithms provide a significant reduction in the total cost of rekeying while ensuring that the number of keys that the users maintain is small. And, if collusion is an issue, then our algorithms provide a significant reduction in the critical cost while ensuring that the total cost of rekeying is manageable.

We note that the problem of user collusion in our solutions is not as serious as that in [5, 6]. In [5, 6], it is possible for two colluding users to compromise *all* the shared keys that the group controller maintains. By contrast, in our solutions, given any two users, they can compromise only a small subset of the shared keys.

## 6. Providing Adaptivity

The proposed combination of logical keys and complementary keys also enables the group controller to

change the algorithm for key distribution at run-time. There are several motivations that can necessitate such a change. We describe some of these motivations next.

**Need to reduce critical cost.** If the changes in application requirements make it necessary to reduce the critical cost during group reorganization (i.e., if the application requires that the interruption during group reorganization be reduced), then the group controller can achieve this by increasing the number of complementary keys used in the algorithm for key distribution. To change the algorithm for key distribution thus, the group controller needs to choose the level at which complementary keys are added. If the group controller chooses to add complementary keys at level  $j$ , then it can determine the cost of such addition by considering the keys used at level  $j+1$ . The cost of such addition can be determined by choosing the appropriate case from Section 3.

For low values of  $j$ , the number of keys that the group controller needs to add is small. However, adding complementary keys at those levels increases the number of decryptions that users need to perform during group reorganization. For high values of  $j$ , the number of subtrees is large and, hence, the number of keys that the group controller needs to add is high. In this case, the group controller can give low priority to the task of generating and sending these keys, as the underlying group communication can continue without them; these keys are useful in reducing the critical cost when a user leaves the group.

**Need to reduce non-critical cost.** If the changes in application requirements make it necessary to reduce the non-critical cost (respectively, the total cost) during group reorganization, then the group controller can achieve this by increasing the proportion of the logical keys. It can achieve this in two ways: (1) increase the logical keys or, (2) remove some complementary keys.

As we can see from the results in Section 3, combining the logical and complementary keys reduces the non-critical cost. More specifically, for the case where both logical and complementary keys are used at level  $j$ , the cost of changing these keys is less than the case where only complementary keys are used at level  $j$  (cf. Section 3.3). Thus, if the users can maintain additional logical keys then the group controller can reduce the non-critical cost by adding logical keys at a level that already uses complementary keys.

The group controller can also reduce the non-critical cost by removing some of the complementary keys. This approach is desirable if the users cannot maintain the additional keys. For removing these keys, it suffices that the group controller does not distribute them during subsequent rekeying.

**Rate of joins/leaves.** If the rate of joins/leaves becomes high, the group controller can choose to use the approaches in Section 5 to reduce the cost of rekey-

ing. While the approaches in Section 5 cannot deal with colluding users, the reduced cost will ensure that the group controller is not overwhelmed due to the number of encryptions it needs to perform (respectively, messages it needs to send). When the rate of joins/leaves is lowered, the group controller can use the technique such as periodic rekeying [7] to ensure that colluding users cannot access the group indefinitely.

Based on the above discussion, it follows that our approach for combining logical and complementary keys allows the group controller to dynamically adapt the algorithm for key distribution based on changing application/user requirements and user capabilities.

## 7. Conclusion

In this paper, we presented a family of algorithms that identified the tradeoff between the critical cost, the cost to change the group key so that the group communication is restored, and the non-critical cost, the cost to change other shared keys so that future changes in group membership are handled quickly. These algorithms are based on logical keys [3] and complementary keys [4].

Based on the application requirements for critical and non-critical cost, the group controller can choose an appropriate algorithm from this family. Moreover, if the application requirements change then the group controller can adapt the algorithm for key distribution accordingly. We showed that for systems where collusion is unlikely or where the rate of joins/leaves is too high, our algorithms can be used to further reduce the total cost of rekeying while ensuring that the number of keys that the users maintain is low. Also, if collusion is likely in a given system then we showed that our algorithms can be used to reduce the critical cost while increasing the total cost slightly.

In the analysis performed in this paper, we focused on the leave operation, as the cost of the critical path is important only for the leave operation. To see this, observe that 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 critical path is crucial in restoring the group communication.

Although we discussed the case where only one user had left, our algorithms can also be used for the case where multiple users leave simultaneously. In such a batch leave, it is possible to reduce the cost of the leave so that the cost of rekeying for the batch is less than the sum of the costs for individual rekeying. We refer the reader to the technical report version of [1] for

the approach of such batch rekeying while using logical and complementary keys.

Our algorithms can be extended to adapt to the users' behavior in a group. If the group controller is aware of profiles of different users in the system then the group controller can change the algorithm for key distribution accordingly. For example, if some users are more likely to leave the group than other users, then these users can be grouped together in one subtree. As an illustration, consider the arrangement shown in Figure 1. If users rooted at  $k_1$  are expected to be transient, the group controller can add a complementary key,  $c_1$ , at that level. However, the corresponding complementary keys  $c_2, c_3, \dots, c_d$  are not added. Now, if the user rooted at  $k_1$  leaves, the group controller can send just one message,  $c_1(k'_g)$ , to the users rooted at  $k_2, k_3, \dots, k_d$ . Thus, these users will receive the new group key quickly. Moreover, the complementary key  $c_1$  need not be changed as the leaving user did not have it. We refer the reader to [8] for work on user profiling.

As discussed in Section 6, our algorithms permit the case where the group controller provides differential level of service to different users. This approach can be extended to deal with the case where the users are heterogeneous in nature and their capabilities/requirements vary. For example, a user with low computing power/low bandwidth may be willing to tolerate a longer delay in obtaining the group key whereas users with high computing power/high bandwidth may want to ensure that the interruption is minimal. It is possible to use our algorithms to provide such differential service.

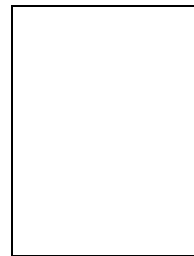
We would like to note that our work differs from that in [9] in that once the group key is established, the group controller need not participate in group communication. By contrast, in [9], several subgroups are maintained and the communication across subgroups is handled by subgroup controllers that decrypt/encrypt messages across subgroup boundary. Also, unlike [7, 10] where the users outside the group can obtain messages sent to a group, our solutions (as well as those in [1–6, 9]) ensure that only current members can obtain messages sent to the group.

## References

- [1] Sandeep Kulkarni and Bezawada Bruhadeshwar. Reducing the cost of the critical path in secure multicast for dynamic groups. In *1st International Workshop on Assurance in Distributed Systems and Networks*, pages 43–48, July 2002. An extended version is available as a technical report MSU-CSE-02-9, Michigan State University.
- [2] H. Harney and C. Muckenhirn. Group key management protocol (GKMP) specification. RFC 2093, July 1997.
- [3] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, 2000.
- [4] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key management for multicast: Issues and architectures. RFC

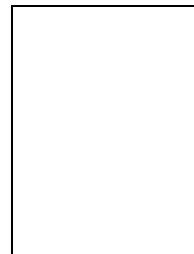
2627, June 1999.

- [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*, 17(9):1614–1631, September 1999.
- [6] Isabella Chang, Robert Engel, Dilip Kandlur, Dimitrios Pendarakis, and Debanjan Saha. Key management for secure internet multicast using boolean function minimization techniques. In *Proceedings IEEE Infocom'99*, volume 2, pages 689–698, March 1999.
- [7] S. Setia, Samir Koushish, and Sushil Jajodia. Kronos: A scalable group re-keying approach for secure multicast. In *IEEE Symposium on Security and Privacy*, pages 215–228, 2000.
- [8] Bruhadeshwar Bezawada. *Secure group communication*. PhD thesis, Michigan State University. in progress.
- [9] S. Mittra. Iolus: A framework for scalable secure multicasting. In *Proc. ACM SIGCOMM'97*, pages 277–288, 1997.
- [10] Michel Abdalla, Yuval Shavitt, and Avishai Wool. Key management for restricted multicast using broadcast encryption. *IEEE/ACM Transactions on Networking*, 8(4):443–454, August 2000.



**Kulkarni** received his B. Tech in Computer Science and Engineering from Indian Institute of Technology, Mumbai, India in 1993. He received his MS and PhD degrees in Computer and Information Science from Ohio State University, Columbus, Ohio, USA in 1994 and 1999 respectively. He has been working as an assistant professor in Michigan State University, East Lansing, USA since August 1999. He is a member of the Software

Engineering and Network Systems (SENS) Laboratory. He is a recipient of the NSF CAREER award. His research interests include fault-tolerance, distributed systems, group communication, security, self-stabilization, compositional design and automated synthesis.



**Bruhadeshwar** received his BE degree in Electronics and Communication Engineering from Osmania University, Hyderabad, India in 1998 and his MS in Electrical Engineering from Michigan State University, East Lansing, USA in 2000. He is currently a PhD student in the Department of Computer Science and Engineering at Michigan State University. His research interests include secure group communication protocols, adaptivity and

distributed algorithms.