Netw Sci (2012) 1: 39–47
DOI 10.1007/s13119-011-0001-8

RESEARCH ARTICLE

Rotation based secure multicast key management for batch rekeying operations

P. Vijayakumar[1,2](✉), S. Bose[1], A. Kannan[3]

1. Department of Computer Science and Engineering, Anna University Chennai, Chennai, Tamilnadu-600025, India
2. Department of Computer Science and Engineering, University College of Engineering Tindivanam, Villupuram, Tamilnadu-604001, India
3. Department of Information Science and Technology, Anna University Chennai, Chennai, Tamilnadu-600025, India

Received: 2 May 2011/Revised: 16 June 2011/Accepted: 16 July 2011

Abstract Many emerging security-rich network applications such as pay-per-view, video broadcasting, video on demand and videoconferencing are based on multicast communication. Thus, securing multicast communications is an important Internet design issue in most of the network applications. In such a scenario providing high security for multicast group members using a common group key is a challenging task. Most of the previous literature describes key tree approaches to distribute the multicast group key in which the rekeying cost is high for batch joining or leaving operations. The marking algorithms proposed in the past focus on batch join and batch leave requests. However, merging and batch balanced algorithms consider batch join more and do not focus much on batch leave operations. In this paper, we present rotation based key tree algorithms to make the tree balanced even when batch leave requests are more than batch joins operations. These proposed algorithms not only maintain a balanced key tree, but also reduce the rekeying costs in comparison with the existing algorithms when batch leave operation is higher than batch join operation (JM < LM). Our simulation result shows that this proposed scheme reduces 20% – 30% rekeying cost compared to the existing approaches.

Keywords rotation algorithms, rekeying operation, group key, balance factor, multicast security

1 Introduction

Multimedia services, such as pay-per-view, videoconferences, some sporting event, audio and video broadcasting are based on multicast communication where multimedia messages are sent to a group. In order to provide security for such multicast communications, the existing systems encrypt the multimedia data using a Group Key and are sent to the set of group members. As group membership is dynamic, this group key is updated and redistributed securely to all group members whenever there is a change in the membership in order to provide forward and backward secrecy. Forward secrecy means that a leaving member cannot obtain information about future group communication and backward secrecy means that a joining member cannot obtain information about past group communication. The operation for updating the group key is known as rekeying. The Group Controller (GC) is responsible for updating the group key when there is a change in the group membership.

Various key management schemes have been proposed in the past [1 – 5] for updating the keys using single rekeying operation in an efficient way. In key tree based approach, each member is assigned a set of keys based on his/her location in the key tree. The rekeying cost of the key tree approach increases logarithmically with the increase in group size for a join or depart request. The rekeying cost denotes the number of messages that need to be distributed to the members in order for them to obtain the new group key. When more number of users join/depart from the multicast service its performance decreases gradually. Group centre and group members are assigned to perform more rekeying operation rather than making use of the service. Batch rekeying [6 – 10] has been proposed to improve these problems. In this scheme, the GC does not perform rekeying

E-mail: vijibond2000@gmail.com

immediately; instead, it consolidates the total number of joining and leaving members during a time interval before performing the rekeying. We refer the algorithm used in [6 – 8] as marking algorithm in this paper. The algorithm used in [9] is referred as merging algorithms and [10] as batch balanced algorithm in this paper. Level homogeneous key tree based key management scheme was proposed [21] to reduce computation and storage complexity. Threshold based rekeying used for wireless channel was also proposed [23] to reduce the communication cost. A key tree is considered balanced if the difference between the heights of its two subtrees is at most 1. For a balanced key tree with N members, the height from the root to any leaf node is $\log_d N$, where d is the maximum number of children of a node of the key tree. If the tree is unbalanced some of the members might need to perform $N-1$ decryptions in order to get the group key. Furthermore, in an unbalanced key tree, some of the members might need to store N keys, whereas remaining members might need to store only few keys.

In this paper, we propose rotation algorithms in which key tree is rotated once or twice in order to make the tree to be balanced when batch leave operations are performed. We call this operation as Rotation once and Rotation twice operations. Rotation once is classified into RL (Rotation Left) and RR (Rotation Right). Rotation twice is also classified into RRL (Rotation in Right and Left) and RLR (Rotation in Left and Right). This rotation based key tree algorithm is more suitable for batch leave operations for rotating the left/right subtrees to balance the tree. These rotation algorithms not only balance the key tree, but have lower rekeying costs compared to the existing algorithms. By reducing the number of rekeying messages required in order to recover the new group key, the computation time is also reduced which is discussed in [17 – 20, 22]. For batch joining operations merging algorithm is more suitable. For same numbers of joining and leaving members, our rotation based algorithms achieve the same rekeying cost as that of the existing algorithms. The rest of the paper is organized as follows: Section 2 discuses the rotation based algorithms and its types. We describe our proposed algorithms in Section 3. Section 4 discuses the simulated and mathematical analysis of our proposed work with existing approaches by comparing the simulated results with them. Section 5 concludes the paper.

2 Rotation algorithms

We now propose two rotation algorithms by redefining the operations of AVL tree [11] to rotate the subtrees in order to maintain the balancing condition. There are few major differences between AVL tree rotation and the rotation operation proposed in this paper. First, AVL is a binary search tree and hence the left node has smaller key than the parent node and right node has greater key value than its parent node. Therefore the levels need not be updated after every rotation. On the other hand in our proposed approach, the tree is only a binary tree and hence the keys are assigned randomly. Moreover, after every rotation any one of the tree traversal (level order) method is to be applied to know the new position of the rotated nodes. This update is monitored and carried out by GC. GC also computes balance factor for each node before rotating the tree. Balance factor of a node is the height of its right subtree minus the height of its left subtree. Thus each node has a balance factor of –1, 0, or +1. Consider balanced key tree which is shown in Fig. 1, in which the balance factors are shown at the top of each node. Balancing factors of leaf nodes are always zero. After computing the balance factors it checks all the nodes whose balance factors are not –1, 0, or +1.

For those nodes rotations are performed in order to make the tree balanced. When a subtree is rotated, the subtree side on which it is rotated decreases its height by one node while the other subtree increases its height by one. This makes it useful for rebalancing a tree. It is not always possible to balance the tree structure in single rotation. Therefore in such cases double rotations are performed in order to balance the tree. Note that rotations are performed from leaf node to the root node.

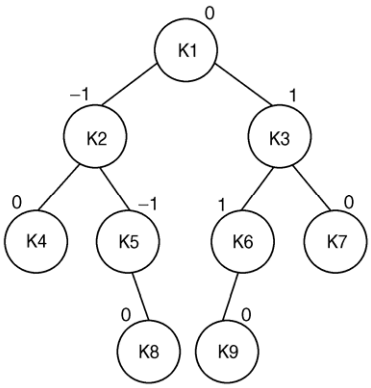

Fig. 1 Balanced key tree

2.1 Rotation once algorithm

Given a batch of join/leave requests, the main task for the GC is to identify which keys should be added, deleted, or changed. In individual rekeying, all the keys on the path from the request location to the root of the key tree have to be changed [12 – 16]. When there are multiple join or leave requests, there are multiple paths to be updated. When such multiple paths are updated, there may be possibilities such that the height of one sub-tree may be higher than the other sub-tree that is the difference between the heights of

the two sub-trees is greater than one. In such scenario, users on one end (sub-tree of height > 1) may be in a situation to perform more rekeying operations comparatively. They also have to store more keys compared with the users who are located on other side of the tree. Hence storage complexity increases for those users. To avoid this situation we introduce two types of rotation algorithms in this section. They are rotation once and rotation twice algorithms.

In rotation once algorithm the key tree is rotated only once to make the tree balanced. Rotation can be done both in left side and right side. When the height of the left subtree is greater than the right subtree height RR rotation is performed. When the height of the right subtree is greater than the left subtree height RL rotation is performed. After rotating the tree, the nodes that have no children are removed from the key tree in order to reduce the rekeying cost.

For example, Fig. 2(a) shows RR rotation in which the key tree is rotated in the right side. After the rotation key node K2 will become an invalid node and the result is shown in Fig. 2(b). In Fig. 2(b) we delete the key node that has no children. This reduces the rekeying cost for batch leave operation. Thus the node K2 is removed from the tree and the resulting tree is shown in Fig. 2(c). Since the resulting tree is an unbalanced tree again the same RR operation is performed and the result is shown in Fig. 2(d). The final result shown in Fig. 2(e) depicts the result of the key tree after deleting the node K1.

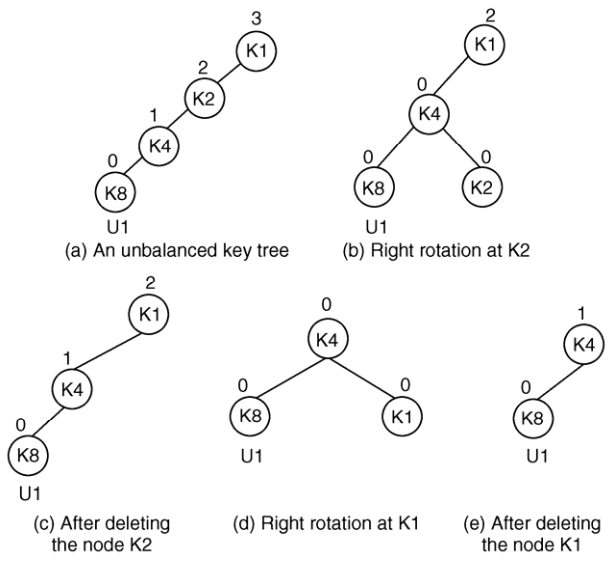

Fig. 2 Example of Rotation Right algorithm

2.2 Rotation twice algorithm

Rotation twice is performed whenever one of the subtree grows in left and right side or right and left side. In such cases, the tree is rotated twice to balance the tree. If the subtree had grown in left and right side the rotations to be performed are RL and RR. The Fig. 3(a) illustrates an example of rotation twice algorithm in which the tree has grown both in left and right side. So the tree is rotated in the left side first. After this rotation, key node K9 becomes the root node for the key node K4. Next the resultant tree is rotated in right direction. The resultant tree is a balanced tree which is shown in Fig. 3(b). In this figure, the node K9 becomes a parent node of K4 and K2. After the rotation if any key node (K4, K2) appears under a user node for example K9, they are simple deleted from the key tree. So the nodes K4 and K2 are deleted and the result is shown in Fig. 3(c)

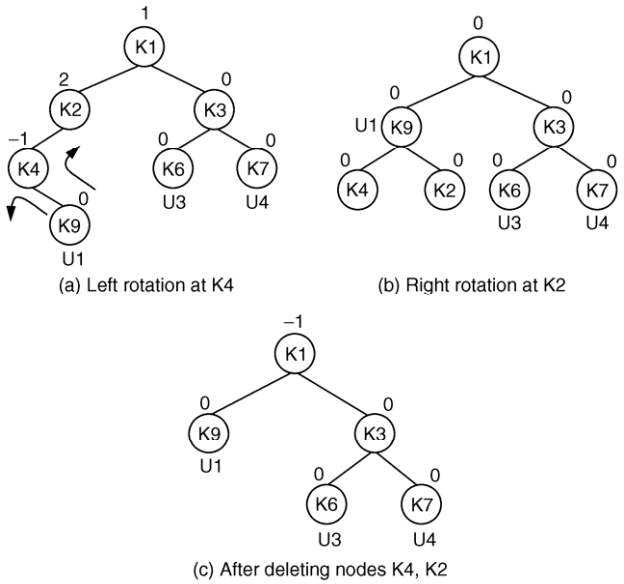

Fig. 3 Example of RLR rotation

3 Rotation based rekeying algorithms

In this paper, we propose rotation based rekeying algorithms which are processed by a GC for batch leave requests. This rotation is applied to the key tree only when the K-key nodes and U-user nodes are organized as a tree. In a key tree, the root is the group key, leaf nodes are individual keys, and the other nodes are auxiliary keys/sub group keys. Rotation based algorithms are not suitable for key star. Key star [1] is a special key tree where tree degree equals to the group size. In both of these methods, the group centre cannot predict which user may leave/join in to multicast service. So it can use a rotation algorithm to make the tree to become a balanced tree in such a way that rekeying operations to be performed by group members are minimized when group of members leave from the service. We use the following notations in this paper.

Notation	Explanation
JM	Number of joining members
LM	Number of leaving members
$\{x\}y$	x is encrypted using y
d	Degree of the tree
h	Height of the tree

There are four cases to be considered for which the rotation based algorithm can be applied. But it is more suitable for the case where LM > JM and JM = 0.

Case 1: if JM = LM then,

(1) Find the nodes in the key tree where leaving operation is going to be performed. Replace all the leaving nodes by joining nodes as discussed in [6 – 8].
(2) All the key nodes are updated form the leaf node to the root node with respect to the leaving point.
(3) Updated rekeying messages are sent to newly established group members.

Case 2: if JM < LM and JM > 0 then,

(1) Find all the nodes where leaving operation is going to take place. Out of the LM leaving nodes pick JM shallowest nodes in the key tree. These selected shallowest nodes are replaced by the newly joining members.
(2) Remaining LM nodes are simply deleted from the key tree.
(3) Compute the balancing factors for all leaf and non- leaf nodes. If the balance factor is not {–1, 1, 0} for any of the nodes of the key tree a suitable rotation operation is to be performed.
(4) Update all the keys from the leaf to root where user leave or join takes place in order to provide forward secrecy.

Consider the key tree in Fig. 4(a) that shows an example of working case 2 (JM < LM) and JM > 0.The figure consists of 16 users starting from user U1 to U16 in which six users (U1, U2, U5, U6, U9, and U10) want to leave the group and two users (U17, U18) want to join the group. Find all the six nodes where leaving operation is going to take place. Out of the six leaving nodes pick two shallowest nodes in the key tree in order to add newly joining members in the key tree. So the locations of the key nodes K23 and K24 are used to add the newly joining members U17 and U18 and all the keys K0, K2, K5, and K11 are changed in order to disallow the newly joining members to view the past transactions. This provides backward secrecy for the new users U17 and U18. The remaining four nodes are simply deleted from the key tree and the result is shown in Fig. 4(b). Since the resultant tree is an unbalanced tree perform appropriate rotations to make the tree to be a balanced tree. The final key tree after the nodes K3, K4, K7 and K9 are removed is shown in Fig. 4(c). After constructing the balance tree the key nodes K0 and K1 are updated in order to stop the users U1, U2, U5, U6 from viewing the future content. This achieves the forward secrecy. It is very clear to see that our proposed rotation based algorithm reduces the rekeying cost from 12 to 10. Moreover it minimizes the number of decryptions to be performed by group members from 34 to 30 which is shown in Table 1.

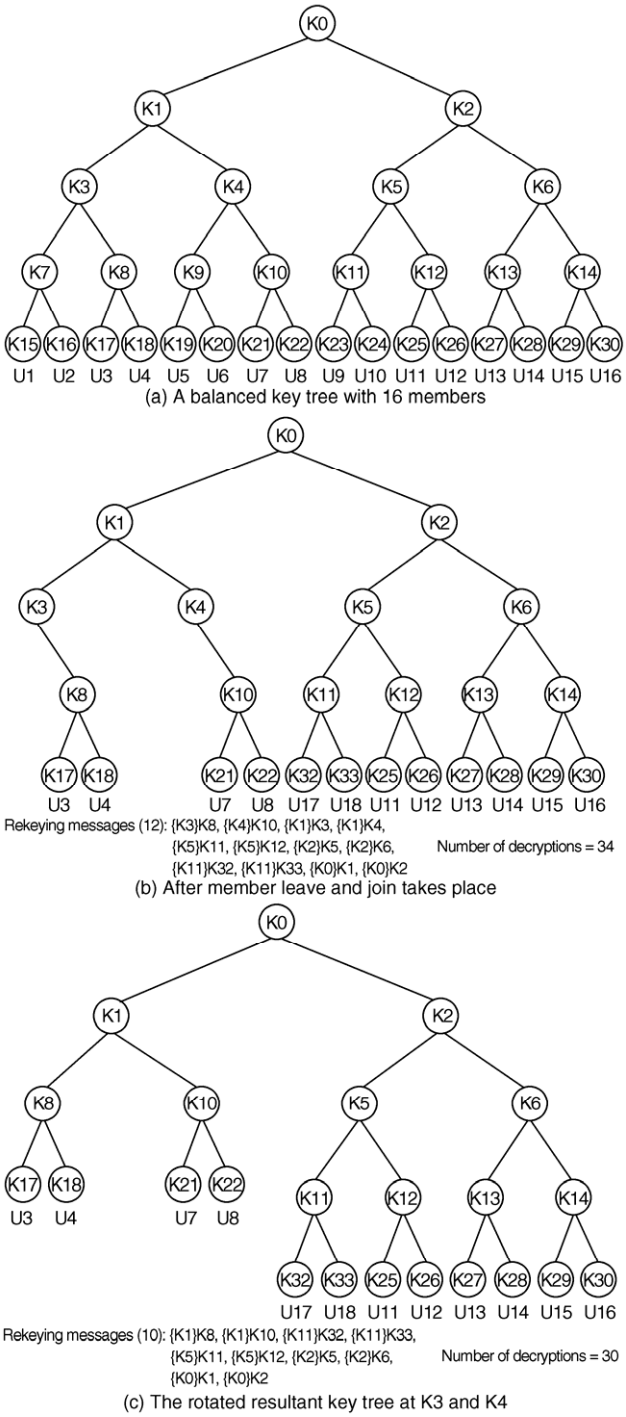

Fig. 4 Working example of the case JM < LM and JM > 0

Table 1 Number of decryptions required in group members area

Users	Number of decryptions performed (without rotation based approach)	Number of decryptions performed in rotation based approach
U3	3	2
U4	3	2
U7	3	2
U8	3	2
U17	4	4
U18	4	4
U11	3	3
U12	3	3
U13 – U16	8	8
TOTAL	**34**	**30**

Case 3: JM > LM and LM > 0 then,

(1) Let the number of joining members be JM and the number of leaving members be LM.
(2) Compute $K = \text{JM} - \text{LM}$, where K represents the number of remaining joining members for whom insertion points are to be found.
(3) Out of JM joining members assign the first LM joining members into the leaving positions in the key tree.
(4) For the remaining "K" joining members, find the insertion point in the key tree. The insertion point is the shallowest node where the join does not increase the height of the key tree. Call this node as insertion node. It may be located either in left or right subtree depending upon where leave/join operations had taken place recently.
(5) At that location group centre creates a new intermediate node, a new member node and promotes the new intermediate node to be the parent of both the insertion node and the new member node.
(6) Compute the balance factor for all the nodes of the key tree.
(7) Perform rotation if necessary.
(8) Update the keying information and send it to group members.

Consider the key tree in Fig. 5(a) that shows an example of working case 3 (JM > LM) and LM > 0. The figure consists of 16 users starting from user U1 to U16 in which two users (U11, U15) want to leave the group and six users (U17, U18, U19, U20, U21, U22) want to join the group. Find the two nodes where leaving operation is going to take place. Select the two leaving points to add the first four newly joining members (U17, U18, U19, and U20) in the key tree. For the remaining two joining members the insertion point is found in such a way that the join does not increase the height of the key tree. It may be located either in left or right subtree depending upon where leave/join operations had taken place recently. Therefore K26 and K30 are selected as insertion points to insert the remaining two users U21 and U22 in the key tree and the result is shown in Fig. 5(b).

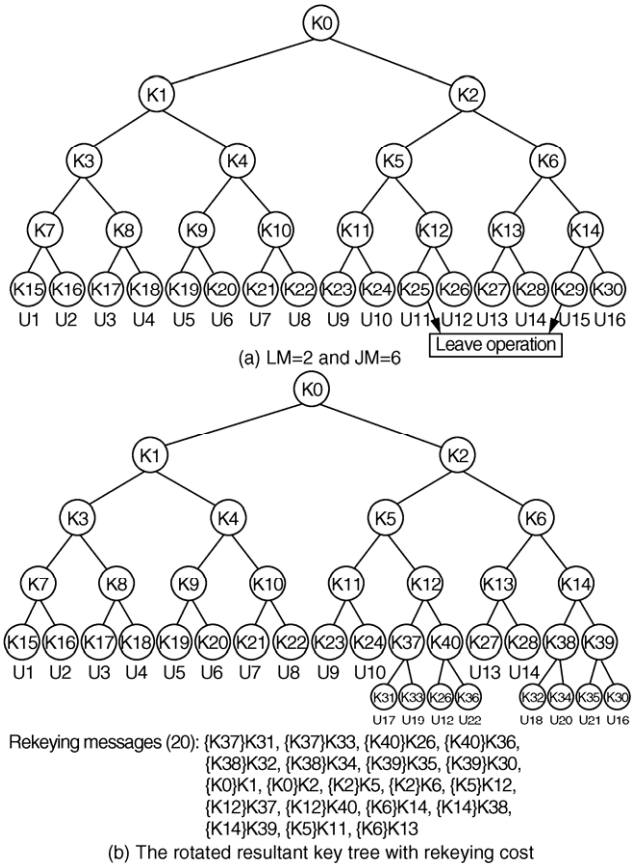

Fig. 5 Working example of the case JM > LM and LM > 0

Case 4: if JM < LM and JM = 0 then,

(1) Delete all the leaf nodes where leave operation had taken place.
(2) Compute the balance factor for all the nodes of the key tree.
(3) If the tree has grown in the left side then do RR.
Else do RL.
(4) If the tree has grown in left and right side simultaneously then do RLR.
Else do RRL.
(5) Perform rekeying operation.

Figure 6(a) shows an example of the working case 4 algorithm. Consider a tree with $d = 2$ and level = 3. Hence the tree can have maximum of 8 members U1 to U8. But for simplicity we have taken only six users in the diagram. In this situation three members U1, U2 and U4 had left the group (LM = 3) and no members would like to join the group (JM = 0).

This structure is shown in Fig. 6(b). This structure does not satisfy the criteria of a balanced tree at K2 and K3. So RL rotation is performed with respect to the nodes K2 and K3 and the result is shown in Fig. 6(c). By doing this rotation

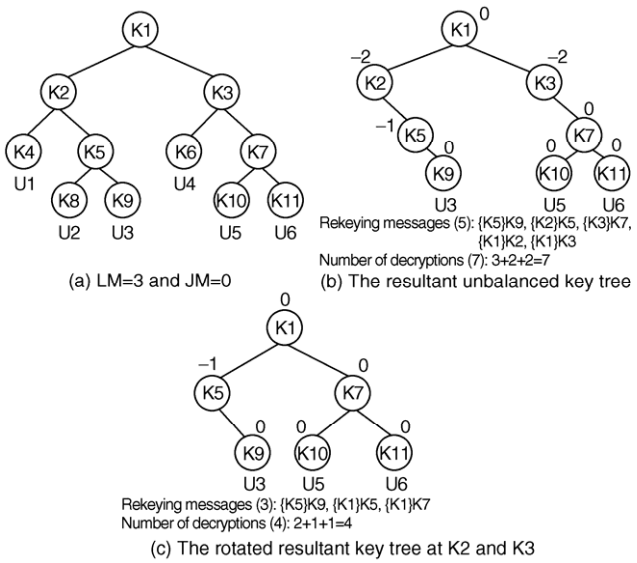

Fig. 6 Working example of the case JM < LM and JM = 0

the rekeying cost is reduced from 5 to 3 and number of decryptions to be performed is reduced from 7 to 4. This improves 40% rekeying cost for the case JM < LM and JM = 0.

Case 5: if JM > LM and LM = 0 then,

(1) Find the insertion point. It is the shallowest leaf node from the left/right subtree.

(2) Create a new intermediate node and a new member node and promote the new intermediate node to be the parent of both the insertion node and the new member node.

(3) Compute the balance factor. Rotate the tree if necessary.

Among all the five cases rotation based algorithms are suitable for case 2 and case 4.

4 Simulation results

In this section, we discuss the performance in terms of rekeying cost of our proposed algorithms and compare them with the marking algorithms described in [6 – 8], merging algorithms [9] and batch balanced algorithms [10] which we have already labeled as Marking, Merging Algorithms and Batch balanced algorithms, respectively. The rekeying cost (RC) denotes the total number of rekey messages that need to be sent to all authorized group members in order for them to find the new group key. After completing the rekeying process, the rekeying messages have to be sent to the remaining group members. These messages should include two information fields: destination node, rekeying material. The destination node is the node to which the message is addressed. This field is used by group members to decide whether the rekeying message concerns to them or not. Rekeying messages indicate the updated keying information which can be used to compute the group key.

Merging algorithms and batch balanced algorithms are more suitable for batch joining requests. However, rotation based rekeying algorithm proposed in this paper is suitable for batch leaving operations. As the number of leaving members increases, more numbers of rotations are performed to balance the tree from unbalanced state. After balancing the tree, rekeying operation is performed and the rekeying cost is analyzed by comparing with marking, merging and batch balanced algorithms. We simulated this approach in Turbo C++ for more than 1024 users with $d = 2$ and compared the results with previous approaches. The simulator first constructs a balanced key tree for 1024 users. Leaving members are either randomly selected or specifically selected so as to give either the best or worst case rekeying costs. Joining members if any are then inserted into the key tree and the rekeying costs are calculated. This shows that our proposed algorithms reduces rekeying costs for the case JM < LM. Figures 7 and 8 show the computed and simulated best and worst cases rekeying costs for a binary key tree with $d = 2$. The graph shown in Fig. 7 depicts the best case rekeying cost results obtained from our simulations.

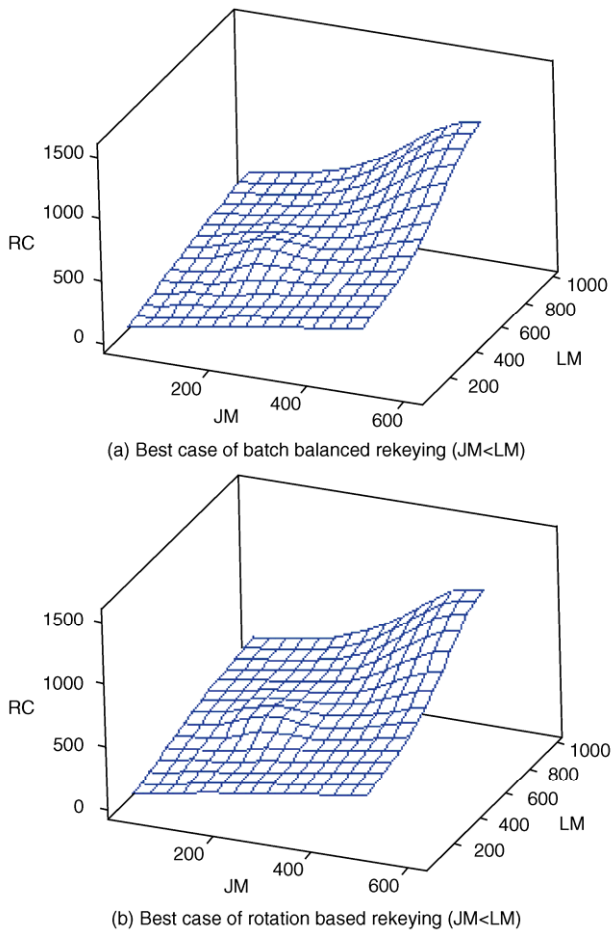

Fig. 7 Batch balanced algorithm rekeying vs. rotation based rekeying (best case)

From Fig. 7(a), it can be seen that batch balanced algorithm has the rekeying cost of 1007 for the total number of JM = 500 and LM = 1000 from a balanced binary tree with 1024 members for the case JM < LM and JM > 0. On the other hand, the rotation based algorithms minimizes the rekeying cost since it try to minimize the number of affected key nodes by rotating the tree and deleting the key nodes that have no children. Figure 7(b) shows that the rotation based algorithm produces the rekeying cost of 998 for the total number of JM = 500 and LM = 1000 from a balanced binary tree with 1024 members. This shows that our proposed algorithms reduce rekeying costs for the case JM < LM and JM > 0.

The graphs shown in Fig. 8 illustrate the worst case analysis of batch balanced algorithm and rotation based rekeying. In worst case analysis the batch balanced algorithm produces the rekeying cost of 1066 for 1024 group members which are shown in Fig. 8(a). In rotation based rekeying, its rekeying cost value is 1054 for the same number of users and the result is shown in Fig. 8(b).

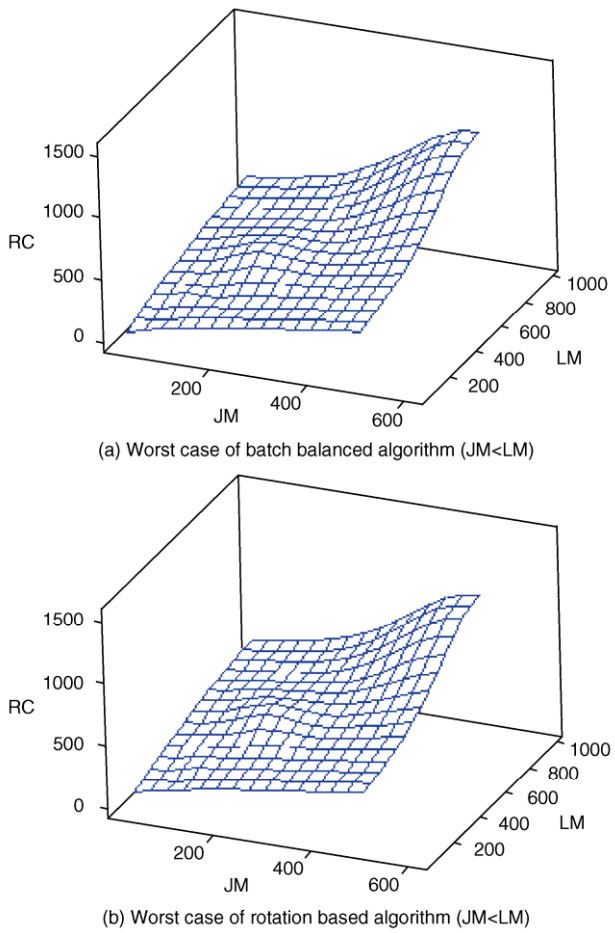

Fig. 8 Batch balanced algorithm rekeying vs. rotation based rekeying (worst case)

In Fig. 9 we compare the worst case analysis of rekeying cost for rotation based algorithms, marking algorithms and batch balanced algorithm for the case (JM < LM and JM = 0). For the worst case analysis of the rekeying cost also, the same balanced tree is constructed with 1024 members of height $h = \log_d N + 1$. The leaving members in such analysis are evenly distributed in the entire sub tree as shown in Fig. 10(a). For the best case analysis of the rekeying cost a balanced tree is constructed with 1024 members of height h and the leaving members concentrate on one area of the sub tree shown in Fig. 10(b). Then the algorithm is run several times approximately 10 runs and the performance is shown below.

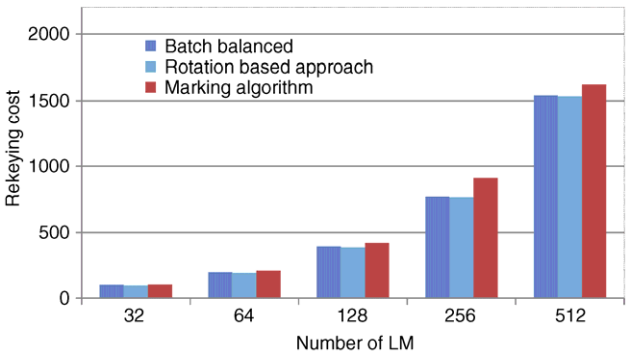

Fig. 9 Worst case rekeying cost comparison for the case LM > JM and JM = 0

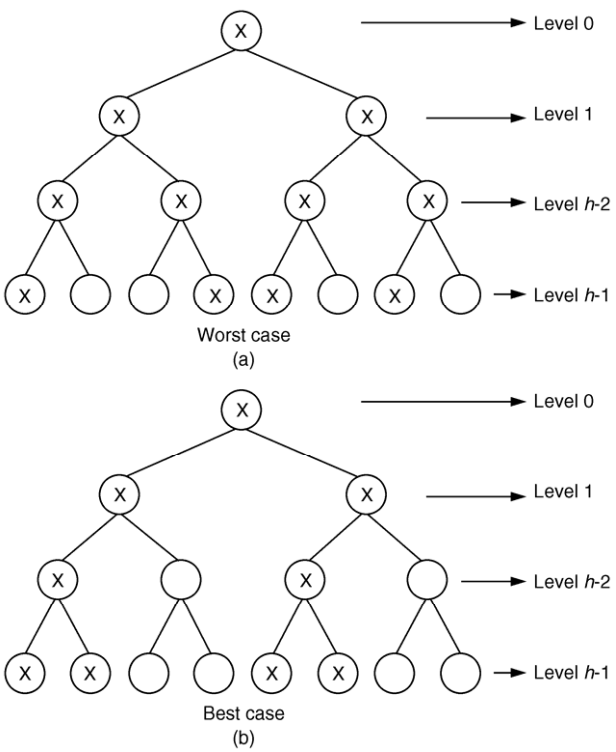

Fig. 10 (a) Worst case and (b) best case scenario

It is clear to see from Fig. 9 that our proposed algorithm minimizes the rekeying cost to 1534 for the LM = 512 and $N = 1024$. Whereas the marking and batch balanced algorithm generates the rekeying cost as 1622, 1540 for the same number of leaving members with JM = 0. The figure shown in Fig. 11, compares best case rekeying cost analysis of rotation based algorithms, marking algorithms and batch balanced algorithm for the case (JM < LM and JM = 0). Then the algorithm is run several times and it is clearly to see that our proposed algorithm minimizes the rekeying cost to 510 for the LM = 512 and $N = 1024$. Whereas the marking and batch balanced algorithm generates the rekeying cost as 520, 516 for the same number of leaving members with JM = 0. We have also performed a theoretical analysis for the rekeying cost of our rotation based rekeying and the details are given below. This analysis has performed both for worst and best cases.

4.1 Worst case analysis

The proposed rotation based algorithms has the ability to control the users join positions. However, they fail to control leaving positions of the users. Thus, worst case analysis mainly considers how the locations of leaves affect the GC cost. In the worst cast analysis all the Leaving members are spread evenly at leaf nodes. The figure shown in Fig. 10(a) illustrates the worst case scenario. During the batch leave operation the numbers of departing members are represented in terms of the degree of the tree. For simplicity we assume that $LM = d^l$ for some integer l. If $LM = d^l + r$, where r value lies between 0 and d^l then for each of the r leaves the rekeying cost decreases gradually in the worst case operation by performing more rotations in the key tree. In order to exhibit the strength of the proposed algorithm, we have calculated WRC (worst case rekeying cost) and BRC (best case rekeying cost) for the batch leave operation. The algorithm is found suitable in both the worst case and best case analysis.

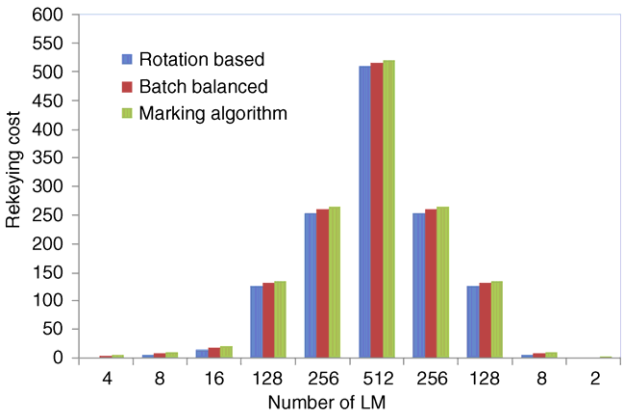

Fig. 11 Best case rekeying cost comparison for the case LM > JM and JM = 0

Case 1: (JM = LM)

$$\text{WRC1} = d\left[\frac{d^{l+1}-1}{d-1}\right] \tag{1}$$

Case 2: (JM > LM)

If $LM = d^l$ for some integer l

$$\text{WRC2} = d\left[\frac{d^{l+1}-1}{d-1}\right] + d[\text{JM} - \text{LM}] \tag{2}$$

Case 3: (JM < LM and JM = 0)

If $LM = d^l$ for some integer l then,

$$\text{WRC3} = d\left[\frac{d^{l}-1}{d-1}\right] + \text{LM}[d-1] \tag{3}$$

Case 4: (JM < LM)

$$\text{WRC4} = d\left[\frac{d^{l+1}-d}{d-1}\right] + [\text{JM} + \text{LM}] \tag{4}$$

4.2 Best case analysis

In the best case analysis all the leaving members are focused only one side (either left or right subtree) of the key tree which is shown in Fig. 10(b). As a result, the rekeying cost will be minimized in comparison to the other existing algorithms. The best case rekeying cost analysis for the case 1 (JM = LM) and case 2 (JM > LM) are the same as that of the discussion done in the worst case analysis. For the remaining cases (3 & 4), consider the number of leaving members $LM = d^l$ where d is the degree of the tree and l is an integer value. In rotation based key tree algorithms, the value of d is considered as 2.

Case 3: (JM < LM) and JM = 0

$$\text{BRC3} = d\left[\frac{d^{l-1}-1}{d-1}\right] \tag{5}$$

Case 4: (JM < LM)

$$\text{BRC4} = d[\log_d N - \log_d \text{LM}] + d[\text{JM} - 1] \tag{6}$$

5 Conclusions

In this paper, a rotation based key tree algorithm has been proposed for reducing the rekeying cost for batch rekeying operation in a multicast group. From the simulations carried out using this algorithm, it has been observed that the proposed algorithm reduces the tree height by performing

rotation operations in the multicast key tree. When the tree is large then this reduces the rekeying cost by 20% – 30% in comparison with the existing approaches. One limitation of this proposed approach is that it works better for batch leave operations are more than batch join operations (JM < LM). When the batch join operations are more than the batch leave operations (JM > LM) the performance is degraded. In those cases, our proposed approach gives the same rekeying cost as that of the existing algorithms. Hence the rotation based algorithms can be combined with merging and batch balanced algorithms to improve the efficiency.

References

[1] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM Trans. Netw.*, vol. 8, no. 1, pp. 16 – 30, Feb. 2000.

[2] W. Trappe, J. Song, R. Poovendran, and K. J. R. Liu, "Key management and distribution for secure multimedia multicast," *IEEE Trans. Multimedia*, vol. 5, no. 4, pp. 544 – 557, Dec. 2003.

[3] P. Vijayakumar, S. Bose, A. Kannan, and S. S. Subramanian, "An effective key distribution protocol for secure multicast communication," in *IEEE Int. Conf. Advanced Computing*, Chennai, India, 2010, pp. 102 – 107.

[4] M. Ramkumar, "The subset keys and identity tickets (SKIT) key distribution scheme," *IEEE Trans. Inf. Forens. Security*, vol. 5, no. 1, pp. 39 – 51, Mar. 2010.

[5] P. Vijayakumar, S. Bose, A. Kannan, and S. S. Subramanian, "A secure key distribution protocol for multicast communication," in *Communications in Computer and Information Science* vol. 140, *Control, Computation and Information Systems*, P. Balasubramaniam, Ed. Heidelberg: Springer, 2011, pp. 249 – 257.

[6] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam, "Batch rekeying for secure group communications," in *Proc. 10th Int. Conf. WWW*, Hong Kong, China, 2001, pp. 525 – 534.

[7] J. Pegueroles and F. Rico-Novella, "Balanced batch LKH: new proposal, implementation and performance evaluation," in *Proc. IEEE Symp. Computers and Communications*, 2003, pp. 815 – 820.

[8] X. B. Zhang, S. S. Lam, D.-Y. Lee, and Y. R. Yang, "Protocol design for scalable and reliable group rekeying," *IEEE/ACM Trans. Netw.*, vol. 11, no. 6, pp. 908 – 922, Dec. 2003.

[9] W. H. D. Ng, H. Cruickshank, and Z. Sun "Scalable balanced batch rekeying for secure group communication," *Comput. Secur.*, vol. 25, no. 4, pp. 265 – 273, Jun. 2006.

[10] W. H. D. Ng, M. Howarth, Z. Sun, and H. Cruickshank, "Dynamic balanced key tree management for secure multicast communications," *IEEE Trans. Comput.*, vol. 56, no. 5, pp. 590 – 605, May 2007.

[11] M. A. Weiss, *Data Structures and Algorithm Analysis in C*, 2nd ed. Boston, USA: Addison-Wesley, 2008, pp. 126 – 139.

[12] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The versakey framework: Versatile group key management," *IEEE J. Sel. Areas Comm.*, vol. 17, no. 9, pp. 1614 – 1631, Sept. 1999.

[13] A. Perrig, D. X. Song, and J. D. Tygar, "ELK: A new protocol for efficient large group key distribution," in *Proc. IEEE Symp. Security and Privacy*, 2001, pp. 247 – 262.

[14] P. P. C. Lee, J. C. S. Lui, and D. K. Y. Yau, "Distributed collaborative key agreement protocols for dynamic peer groups," in *IEEE Int. Conf. Network Protocols (ICNP)*, Paris, France, 2002, pp. 322 – 333.

[15] L. Xu and C. Huang,"Computation-efficient multicast key distribution," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 5, pp. 1 – 10, May 2008.

[16] B. Bruhadeshwar and S. S. Kulkarni, "Balancing revocation and storage trade-offs in secure group communication," *IEEE Trans. Depend. Secure Comput.*, vol. 8, no. 1, pp. 58 – 73, Jan. 2011.

[17] B. Bruhadeshwar and K. Kothapalli, "A family of collusion resistant symmetric key protocols for authentication," in *Proc. 9th Int. Conf. Distributed Computing and Networking*, Kolkata, India, 2008, pp. 387 – 392.

[18] B. Bruhadeshwar, K. Kothapalli, and M. S. Deepya, "Reducing the cost of session key establishment," in *Proc. 6th Int. Conf. Availability, Reliability and Security*, Fukuoka, Japan, 2009, pp. 369 – 373.

[19] B. Bruhadeshwar, K. Kothapalli, M. Poornima, and M. Divya, "Routing protocol security using symmetric key based techniques," in *Proc. 6th Int. Conf. Availability, Reliability and Security*, Fukuoka, Japan, 2009, pp. 193 – 200.

[20] S. S. Kulkarni and B. Bruhadeshwar, "Key-update distribution in secure group communication," *Comput. Commun.*, vol. 33, no. 6, pp. 689 – 705, Apr. 2010.

[21] D.-H. Je, J.-S. Lee, Y. Park, and S.-W. Seo, "Computation-and-storage efficient key tree management protocol for secure multicast communications," *Comput. Commun.*, vol. 33, no. 2, pp. 136 – 148, Feb. 2010.

[22] A. T. Sherman and D. A. McGrew, "Key establishment in large dynamic groups using one-way function trees," *IEEE Trans. Softw. Eng.*, vol. 29, no. 5, pp. 444 – 458, May 2003.

[23] J. H. Cho, I.-R. Chen, and M. Eltoweissy, "On optimal batch rekeying for secure group communications in wireless networks," *Wireless Netw.*, vol. 14, no. 6, pp. 915 – 927, Dec. 2008.