Revised Variable Length Interval Batch Rekeying with Balanced Key Tree Management for Secure Multicast Communications

Joe Prathap, P.M[†] and Vasudevan, V[†]

Department of Information Technology,
[†]Arulmigu Kalasalingam Coll. of Engg., KrishnanKoil, 626 190, India.

Summary
With the evolution of the Internet, multicast communications seem particularly well adapted for large scale commercial distribution applications, for example, the pay TV channels and secure videoconferencing. A key tree approach has been proposed by other authors to distribute the multicast group key in such a way that the rekeying cost scales with the logarithm of the group size for a join or depart request. The efficiency of this key tree approach critically depends on whether the key tree remains balanced over time as members join or depart. So the researchers try to create a balanced tree by applying merging algorithms for batch join requests and to handle the batch depart request they extended and created a batch balanced algorithm. But we found that the algorithm works well only if the number of joining members is greater than the number of departing members. In this paper we analyzed various strategies and extended the Batch balanced algorithm further by utilizing variable length batch rekeying interval. This paper analyses the efficiency of the proposed scheme with the existing schemes and the comparison shows that the proposed scheme performs better than the existing schemes in terms of balanced key tree generation and minimizing the number of key update messages.

Keywords : Multicast security, group key management, secure group communication, rekeying

1 INTRODUCTION

Multicasting is a type of communication between computers in a network that enables a computer to send one stream of data to many interested receivers without interrupting computers that are not interested. For these reasons, multicasting has become the favored transmission method for most multimedia and triple play applications, which are typically large and use up a lot of bandwidth. Multicasting not only optimizes the performance of your network, but also provides enhanced efficiency by controlling the traffic on your network and reducing the loads on network devices. This technology benefits many group communication applications such as pay-per-view, online teaching, and share quotes [4], [6].

Before these group oriented multicast applications can be successfully deployed, access control mechanisms [7], [9], [13], [22] must be developed such that only authorized members can access the group communication. The only way to ensure controlled access to data is to use a shared group key, known only to the authorized members, to encrypt the multicast data. As group membership might be dynamic, this group key has to be updated and redistributed securely to all authorized members whenever there is a change in the membership in order to provide forward and backward secrecy [5] [8]. Forward secrecy means that a departing member cannot obtain information about future group communication and backward secrecy means that a joining member cannot obtain information about past group communication. We assume the existence of a trusted entity, known as the Group Controller (GC), which is responsible for updating the group key. This allows the group membership to scale to large groups. A number of scalable approaches have been proposed and one in particular, the key tree approach [2], [3], [10], [20], [23], [24], is analyzed in detail and extended in this paper. In short, the key tree approach employs a hierarchy of keys in which each member is assigned a set of keys based on its location in the key tree. The rekeying cost of the key tree approach increases with the logarithm of the group size for a join or depart request [16], [17], [18]. The operation for updating the group key is known as rekeying and the rekeying cost denotes the number of messages that need to be disseminated to the members in order for them to obtain the new group key.

Individual rekeying, that is, rekeying after each join or depart request, has two drawbacks [12], [14],[18]. First, it is inefficient since each rekey message has to be signed for authentication purposes and a high rate of join/depart requests may result in performance degradation because the signing operation is computationally expensive. Second, if the delay in a rekey message delivery is high or the rate of join/ depart requests is high, a member may need a large amount of memory to temporarily store the rekey and data messages before they are decrypted. Batch rekeying techniques have been recently presented as a solution to overcome this problem. In such methods, a departed user will remain in the group longer and a new user has to wait longer to be accepted. All join and leave requests received within a batch period are processed together at the same time. A short rekey interval does not provide much batch rekeying benefit, whereas a long rekey interval causes a delay to joining members and increases vulnerability from departing members who can still receive the data.

The efficiency of the key tree approach critically depends on whether the key tree is balanced [21], [24], [25]. A key tree is considered balanced if the distance from the root to any two leaf nodes differs by not more than 1. For a balanced key tree with N members, the height from the root to any leaf node is $\log_k N$, where k is the out degree of the key tree, but, if the key tree becomes unbalanced, then the distance from the root to a leaf node can become as high as N. In other words, this means that a member might need to perform N - 1 decryptions in order to get the group key.

Recently, two Merging Algorithms suitable for batch join events for combining subtrees together was proposed [18]. These two Merging Algorithms not only balance the key tree, but have lower rekeying costs compared to existing algorithms. In order to additionally handle departing members, the above algorithm extend to a Batch Balanced Algorithm [1] where the tree height adapts to the change in the group membership. However, this requires a reorganization of the group members in the key tree. But this Batch Balanced Algorithm performs significantly better than existing algorithms only when the number of joining members is greater than the number of departing members or when the number of departing members is around N=k, with no joining members. Our approach extends this algorithm further by using two phase batch rekeying interval. This will try to avoid the number of departing members exceeds the number of joining members as much as possible. And in turn lead to improve the overall performance when compared with existing work.

2 BACKGROUND

2.1 Key Tree Approach

In a typical key tree approach [3], [20], [23] as shown in Fig. 1a, there are three different types of keys: Traffic Encryption Key (TEK), Key Encryption Key (KEK), and individual key. The TEK is also known as the group key and is used to encrypt multicast data. To provide a scalable rekeying, the key tree approach makes use of KEKs so that the rekeying cost increases logarithmically with the group size for a join or depart request. An individual key serves the same function as KEK, except that it is shared only by the GC and an individual member.

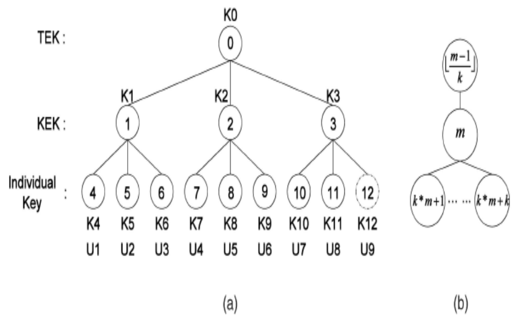

TEK :
KEK :
Individual Key :
K0
0
K1
K2
K3
1
2
3
4
5
6
7
8
9
10
11
12
K4 K5 K6 K7 K8 K9 K10 K11 K12
U1 U2 U3 U4 U5 U6 U7 U8 U9
(a)
$\lfloor \frac{m-1}{k} \rfloor$
m
k*m+1
k*m+k
(b)

Fig 1.(a) key tree structure (b)ID assignment

In the example in Fig. 1a, K0 is the TEK, K1 to K3 are the KEKs, and K4 to K12 are the individual keys. The keys that a group member needs to store are based on its location in the key tree; in other words, each member needs to store $1+\log_k N$ keys when the key tree is balanced. For example, in Fig. 1a member U1 knows K0, K1, and K4 and member U7 knows K0, K3, and K10. The GC needs to store all of the keys in the key tree.

To uniquely identify each key, the GC assigns an ID to each node in the key tree. The assignment of the ID is based on a top-down and left-right order. The root has the lowest ID, which is 0. For a node with an ID of m, its parent node has an ID of (m-1)/k], with its children's IDs ranging from km+1 to km+k, as shown in Fig. 1b.

When a member is removed from the group, the GC must change all the keys in the path from this member's leaf node to the root to achieve forward secrecy. All the members that remain in the group must update their keys accordingly. If the key tree is balanced, the rekeying cost for a single departing member is $k\log_k(N)-1$ messages. For example, suppose member U9 is departing in Fig. 1a. Then, all the keys that it stores (K0 and K3) must be changed, except for its individual key.

If backward secrecy is required, then a join operation is similar to a depart operation in that the keys that the joining member receives must be different from the keys previously used in the group. The rekeying cost for a single joining member is $2\log_k N$ messages when the key tree is balanced.

The efficiency of the key tree approach critically depends on whether the key tree remains balanced. For a balanced key tree with N leaf nodes, the height from the root to the any leaf node is $\log_k N$. However, if the key tree becomes unbalanced, the distance from the root to a leaf node can become as high as N. and if it is unbalanced we can't predict the number of rekeying messages also.

2.2. Batch Rekeying Approach

Before we proceed our work, we introduce some notations and definitions used in this paper. We use "minimum height" to mean the minimum number of levels in a tree or subtree from the root to any leaf node. We define the following variables:

ST	Sub Tree
J	Number of joining members
D	Number of departing members
h	Height of key tree($1+\log_k N$)
H_{MIN}	Minimum height of the leaf node
H_{MAX}	Maximum height of the leaf node
H_{INSERT}	H_{MIN} of ST_A-H_{MAX} of ST_B
BI_{MIN}	Minimum Batch rekeying Interval
BI_{MAX}	Maximum Batch rekeying Interval
BRI	Batch rekeying interval

Marking Algorithms have been proposed to update the key tree and generate, at the end of each rekey interval, a rekey subtree with a collection of join and depart requests. Several variations of Marking Algorithms have been proposed [17], [18]. We refer to the algorithm in [17] as Marking Algorithm 1. For this algorithm, there are four cases to consider. If J=D, then all departing members are replaced by the joining members. If J < D, then we pick the J shallowest leaf nodes from the departing members and replace them with the joining members. If J >D and D=0, then the shallowest leaf node is selected and removed. This leaf node and the joining members form a new key tree that is then inserted at the old location of the shallowest leaf node. Next, if J >D and D > 0, then all departing members are replaced by the joining members. The shallowest leaf node is selected from these replacements and removed from the key tree. This leaf node and the extra joining members form a new key tree that is then inserted at the old location of the removed leaf node. Last, the GC generates the necessary keys and distributes them to the members.

The algorithm in [18] is referred to here as Marking Algorithm 2. There are only three cases to consider for this Marking Algorithm. Two of them, J=D and J<D, are similar to the one mentioned above, except that the nodes of departing members that are not replaced by the joining members are marked as null nodes. For J>D, all departing members are replaced by the joining members. If there are null leaf nodes in the key tree, then they are also replaced by the joining members, starting from the null nodes with the smallest node ID. If there are still extra joining members, then the member with the smallest node ID is removed and it is inserted as a child, together with k-1 joining members at its old location. The next smallest node ID member is selected if there are more joining members. This insertion continues until all of the joining members have been inserted into the key tree. As before, the GC distributes the new key to the members.

3 BATCH REKEYING ALGORITHM

We now propose two Merging Algorithms to combine subtrees together in a way that is suitable for batch join events. To handle all cases such as depart or both join and depart requests, we then extend these two Merging Algorithms into a Batch Balanced Algorithm. The two Merging Algorithms are used to combine two subtrees: ST_A and ST_B. We assume that ST_A has a greater height than ST_B and both subtrees are of the same out degree k.

3.1 Merging Algorithm 1

This algorithm is only used when the difference in the maximum height between the two subtrees ST_A and ST_B is greater than or equal to 1.We now describe Merging Algorithm 1. The criteria for choosing Merging Algorithm 1 is when the difference between $H_{MAX\_ ST\_ A}$ and $H_{MIN\_ ST\_ B}$ is greater than 1 and when the difference between $H_{MAX\_ ST\_ A}$ and $H_{MAX\_ ST\_ B}$ is greater than or equal to 1. If both of these conditions are fulfilled, then the algorithm calculates H_{INSERT}. The following steps are then performed:

Step 1. For k > 2, the algorithm searches for an empty child node in ST_A at either level H_{INSERT} or level H_{INSERT}-1. If H_{INSERT}=0, then levels 0 and 1 are searched. If such a node exists, then the algorithm inserts ST_B as the child of that particular key node.

Step 2. If an empty node is not found in Step 1, mark a suitable key node in ST_A at level H_{INSERT} for insertion as follows: If H_{INSERT} =0, then a suitable key node at level 1 is marked. The marked key node is given by the one with the greatest number of leaf nodes at level $H_{MIN\_ST\_A}$.

Step 3. For k > 2, when an empty node is not found in Step 1, the algorithm searches the root of ST_B for an empty node. If this exists, then the algorithm inserts the marked key node from Step 2 as the child of ST_B and inserts ST_B at the old location of the marked key node.

Step 4. For k = 2 or k > 2, if Steps 1 to 3 have not inserted ST_B into ST_A, then the algorithm creates a new key node at the old location of the marked key node (from Step 2) and inserts the marked key node and ST_B as its children.

Finally, the GC may need to multicast at most one update message to inform the affected members.

3.2 Merging Algorithm 2

We now describe our Merging Algorithm 2 [18]. This algorithm is only used for combining subtrees whose height difference is 0 or equal to 1. The criteria for using Merging Algorithm 2 are when the difference between $H_{MAX\_ST\_A}$ and both $H_{MIN\_ST\_B}$ and $H_{MAX\_ST\_B}$

is 0 or equal to 1. The algorithm performs the following steps

Step 1. For k > 2, the algorithm searches the root of ST_A for an empty child key node. If it exists, then the algorithm inserts ST_B at the empty child key node.

Step 2. For k = 2 or when Step 1 is not valid for k > 2, the algorithm creates a new key node at the root and inserts ST_A and ST_B as its children.

The GC needs to multicast at most one update message to all existing members. After updating the affected node IDs, the members can identify the set of keys that they need in the rekey messages.

3.3 Batch Balanced Algorithm

We now show how our two Merging Algorithms can be extended to produce an algorithm that we call Batch Balanced Algorithm that encompasses both joining and departing members.

There are six steps in our Batch Balanced Algorithm.

1. Identify and mark all key nodes that need to be updated. These key nodes are on the ancestor paths from each departing member to the root.
2. Remove all marked key nodes. After removal, there are only two types of element left: the remaining subtrees and the joining members.
3. Classify all siblings of the departing members as joining members since all of the KEKs that they store cannot be used.
4. Group the joining members into one or many subtrees, each with k members. If there are remaining members left, then they are grouped into another subtree of between 2 and k - 1 members unless there is only one member left. If there is only one member left, then treat it as a single-node subtree.
5. Starting from the subtree with the minimum height, compare it with another subtree with the next minimum height and if the Merging Algorithm 1 criteria are met, combine them using Merging Algorithm 1, else combine them using Merging Algorithm 2. Repeat this process until there is only one key tree.
6. Construct the update and rekey messages and multicast them to the members.

For clarity, we illustrate it with an example. Assume that we have a key tree with 16 members. Suppose members U11 and U15 are departing from the group and six new members, U17 to U22, are joining the group.

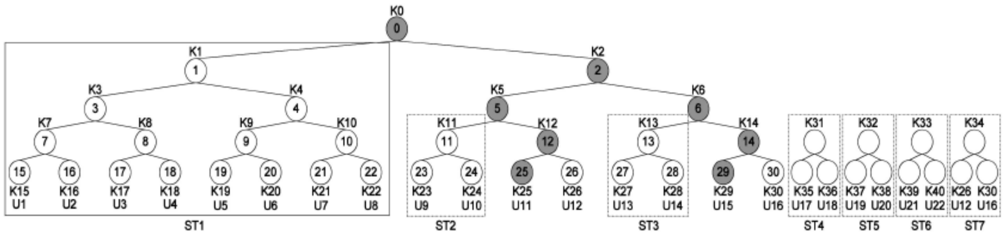
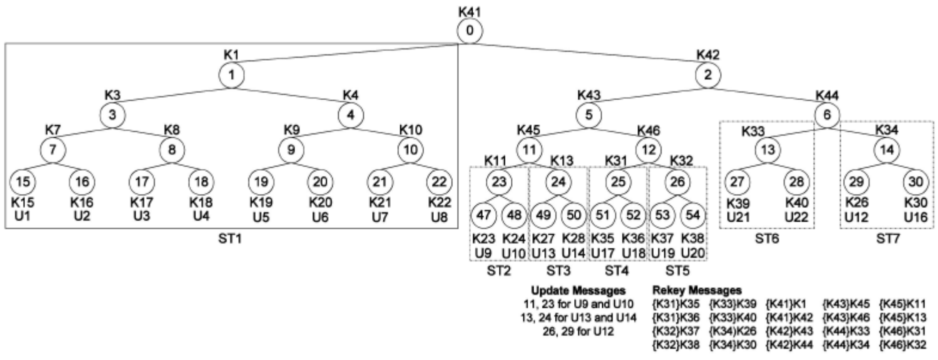


Fig. 2. Steps 1 to 4 of the Batch Balanced Algorithm.



Update Messages
11, 23 for U9 and U10
13, 24 for U13 and U14
26, 29 for U12

Rekey Messages

{K31}K35	{K33}K39	{K41}K1	{K43}K45	{K45}K11
{K31}K36	{K33}K40	{K41}K42	{K43}K46	{K45}K13
{K32}K37	{K34}K26	{K42}K43	{K44}K33	{K46}K31
{K32}K38	{K34}K30	{K42}K44	{K44}K34	{K46}K32

Fig. 3. Resulting key tree.

All of the key nodes in the path from the departing members to the root are marked and removed (Steps 1 and 2). The siblings of departing members U12 and U16 form a new subtree, ST7, since the KEKs that they store are unusable (Step 3). The joining members form one or more subtrees of k members (Step 4). These usable subtrees ST1 to ST7 are identified as shown in Fig. 2.In Step 5, we start with the minimum-height subtrees and merge them. Thus, ST2 forms a subtree with ST3, ST4 forms a subtree with ST5, and ST6 forms a subtree with ST7. Then, the resulting subtree of ST2 and ST3 is combined with the resulting subtree of ST4 and ST5.

This resulting subtree, in turn, forms another subtree with the resulting subtree of ST6 and ST7. Finally, the last two subtrees form a single key tree, as shown in Fig. 3. The GC sends out the update messages to inform the members of their new location. Those members that need to receive the update messages are U12 and the members in ST2 and ST3, which means that a total of three update messages is needed. In this example, we assume that member U16 and subtree ST1 are left intact at their old location. If their locations are changed, then two extra update messages are needed. For ST4, ST5, and ST6, no update message is needed since the members in the subtrees are newly joining members. At the same time, the GC can multicast the rekey messages to the members. The total rekeying cost is 20 messages.
If we use Marking Algorithm 1 [17] or Marking Algorithm 2 [18] in a similar situation, then Marking Algorithm 1 has the same rekeying cost, but it ends up with an unbalanced key tree. Although Marking Algorithm 2 can maintain a balanced key tree, it needs 28 rekey messages. From this, we can see that reorganizing the group members leads to saving on rekeying costs.

4 REVISED TWO PHASE BATCH REKEYING ALGORITHM

The batch rekeying with variable interval is more suitable to the network than that with fix interval, because the batch rekeying with variable interval leads to the steady rekey traffic and cost of rekey [15]. Keeping this point in mind, we are going to apply variable batch rekey interval for rekeying. It has two threshold interval level; lower threshold called as **BI_{MIN}**, which means minimum interval at which rekeying can occur. Higher threshold called as **BI_{MAX}**, which means maximum batch rekey interval. The exact batch rekey interval will be called as **BRI** , and in the range of,

$$\mathbf{BI_{MIN} \leq BRI \geq BI_{MAX}} \qquad (1)$$

Based on the multicast application & its required security level, we can choose the threshold limits **BI_{MIN}** & **BI_{MAX}**. Its operation as follows: the current batch rekey interval was chosen based on the following condition. The algorithm will wait until the minimum batch interval **BI_{MIN}** to occur. After reaching the time interval **BI_{MIN}**, now the algorithm checks whether J >= D condition will achieved or not. If the condition satisfied then **BI_{MIN}** will be considered as the Batch rekey interval **BRI.** If the condition will not occur then for each join or depart request the system will continuously check if the J>=D condition achieved or not. And if it happened means that particular current time will be taken as the current **BRI.** But sometimes the condition J >= D will not occur for a long period of time and it will reach the maximum batch rekey limit **BI_{MAX}**. Then **BI_{MAX}** will be considered as the current **BRI**. Thus we are trying to avoid the condition J < D as much as possible, and the performance of the algorithm improved further. After finding the **BRI** the group controller will apply the batch balanced algorithm. The above possibilities will be explained by various cases shown below.

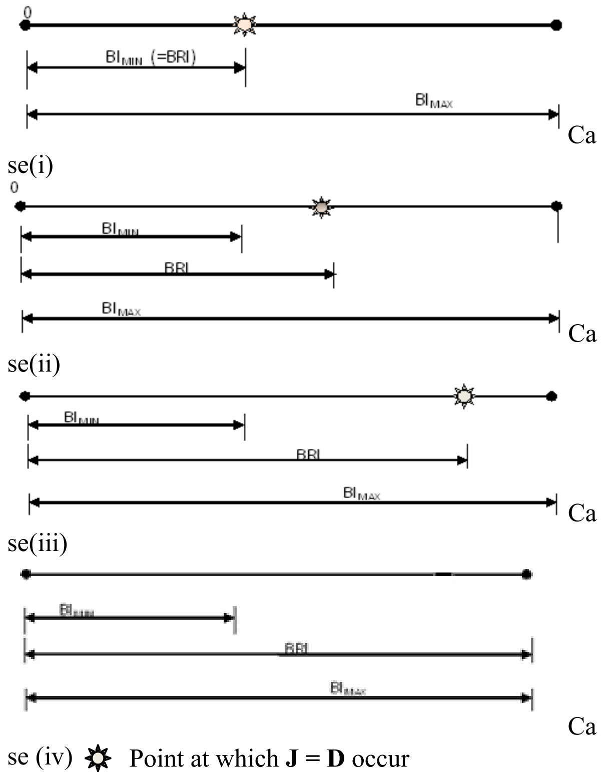


Fig 4. Various possibilities for calculating **BRI**

In fig 4, various cases for calculating batch rekeying interval was depicted and each cases explained below. In Case(i) the group will receive the join requests and depart requests until it receive the **BI_{MIN}**. When reaching **BI_{MIN}** it will compare the total number of join and depart requests. If it found that J > D or J =D then the **BI_{MIN}** considered as a batch rekey interval and the batch balanced algorithm applied. In case (ii) & (iii), the group will receive the join requests and depart requests upto **BI_{MIN}**, and check whether J >= D condition occur or not and found that it will not happened. So it continuously receive join & depart requests and check for the same condition. If it happened then that particular moment will be considered as the current **BIR** and the batch balanced algorithm applied. In case (iv) also the above conditions applied. But the condition J >= D will never occur within the threshold limits. So the **BI_{MAX}** considered as the current batch rekey interval **BIR** and the batch balanced algorithm applied. To determine the batch interval we should consider two factors: the average delay of users request response and the batch traffic which is determine by the number of users request in batch interval. Also, while choosing the threshold limits $\mathbf{BI_{MIN}}$ & $\mathbf{BI_{MAX}}$ care should be taken so that we can preserve the forward and backward secrecy optimally.

4.1. Update Messages

In order for the members to identify the keys that they need after the key tree has been reorganized, the GC needs to inform the members of their new location [11]. An update message consists of the smallest node ID of the usable key tree m and the new node ID m'. With the new node ID m', the members can update the remaining keys m_0 by using the following function:

$$f(m_0)= K^{x}(m'-m)+ m_0 \qquad (2)$$

where x denotes the level of the usable key tree.

5. PERFORMANCE EVALUATION

In this section, we study the performance of our proposed algorithms and compare them with the existing batch balanced Algorithm. We consider four performance metrics:

- rekeying cost,
- update cost,
- minimum and maximum height in the key tree, and
- Key storage.

The rekeying cost denotes the total number of rekey messages that need to be sent to all authorized group members in order for them to learn the new group key. A higher rekeying cost means that more bandwidth is needed for the transmission. The update cost denotes the total number of update messages that need to be sent to all affected members after the key tree has been reorganized in order for them to identify the keys that they need. As for the minimum and maximum height, they affect the members' key storage and, thus, the number of decryptions needed by each member and may even increase the rekeying costs, too. Last, the key storage denotes the number of keys each member need to store.

We ran our algorithms on a Linux terminal with a 512 Mbyte RAM on a 2 GHz processor. To give an indication of runtime, for a tree size of 4,096 members, runtimes are typically in the range of 1 to 5 sec and, for a tree size of 65,536 members, runtimes are typically in the range of 1 to 40 sec, both results being less than or equal to approximately 2,000 departing and joining members.

5.1. Batch Balanced Algorithm

5.1.1 Rekeying Cost

A theoretical analysis for the rekeying cost of the Batch Balanced Algorithm was already done in [1].we have built a simulator for the algorithm. The simulator first constructs a balanced key tree with 1,024 members for k = 2. Departing members are either randomly selected. Joining members are then inserted into the key tree and the rekeying costs are calculated.

The theoretical analysis and the simulated results match so well that we could not distinguish between the two. The highest rekeying cost occurs when the number of departing members approaches half the group size, which means that most or all the key nodes in the key tree cannot be used.

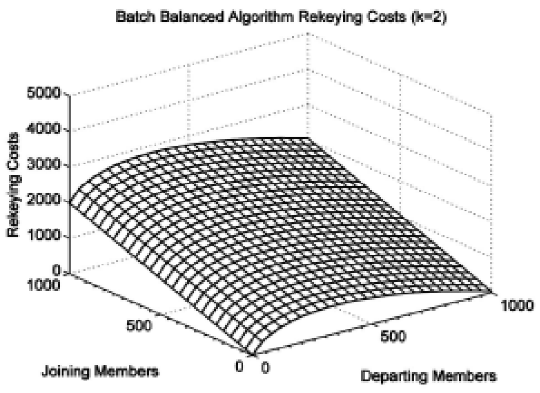


Fig.5. Rekeying costs for Batch Balanced Algorithm.

If the departing members are randomly selected, then we obtain the mean rekeying costs that lie between the theoretical best and worst cases[1]. Generally, we can predict the rekeying costs for a key tree of any outdegree k if we are able to group the members according to their departing probability since it is based purely on the number of joining members rather than the number of departing members. However, if the departing members are spread around as in the worst case, the highest rekeying cost happens when the number of departing members is around N=k since most or all of the KEKs that the members store cannot be used.

Fig. 5 shows the rekeying costs for the batch balanced algorithm for k = 2.

5.1.2 Update Cost

For the Batch Balanced Algorithm, there are some overheads incurred since we reorganize the group members in the key tree.

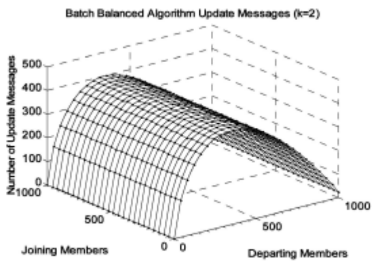


Fig. 6. Update messages for the Batch Balanced Algorithm (k=2).

Fig. 6 shows the total update messages that need to be sent to the remaining group members, including the siblings of the departing members, in order for them to update their new key node IDs. As expected, the update messages are purely dependent on the number of departing members. The number of update messages increases as the number of departing members increases to around half the group size. This is because more key nodes in the key tree are affected by the departing members. However, once the number of departing members exceeds half the group size, the number of update messages decreases since there are fewer members left in the group. If we assume that a key is 128 bits long and the node ID is 20 bits (that is, up to 220 members), then a rekey message is at least 148 bits, excluding other overheads. An update message consists of the old node ID and the new node ID and, ignoring overheads, is therefore 40 bits long. In other words, a rekey message is 3.7 times the length of an update message; thus, the maximum update cost is equivalent to 109 rekey messages. Fig. 7 shows the total number of update messages that need to be multicast to the members for k = 4.

We can see that there is a sharp increase in update messages compared with a binary key tree. This is because, for every departing member, the GC needs to send three update messages to its siblings so that they can update the new location. The highest number of update messages occurs when the number of departing members is in the region of N/k.

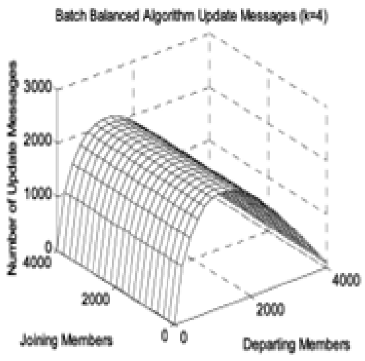


Fig. 7. Update messages for the Batch Balanced Algorithm (k=4).

5.1.3 Minimum and Maximum Height

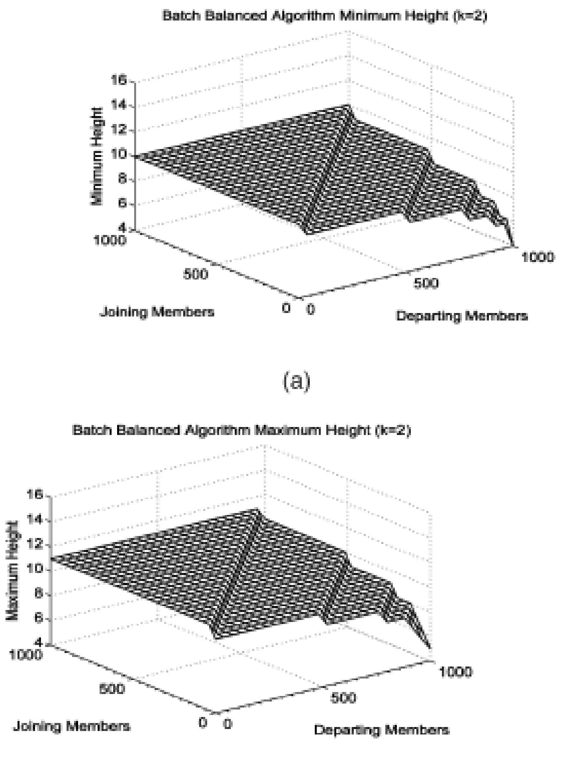
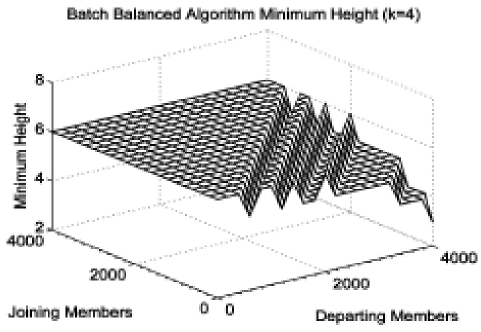
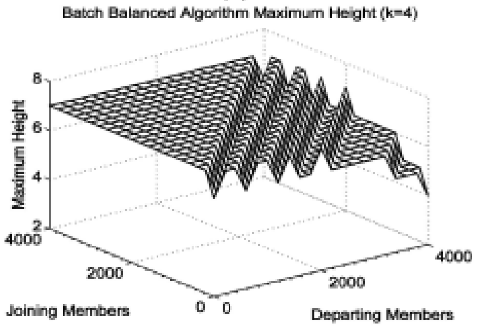


Fig. 8. (a) Minimum and (b) maximum height for the Batch Balanced Algorithm (k=2)

Fig. 8. shows the minimum and maximum heights for the Batch Balanced Algorithm. Regardless of the number of joining or departing numbers, both minimum and maximum height adapt to the changes in the group membership.



(a)



(b)

Fig. 9. (a) Minimum and (b) maximum height in the Batch Balanced Algorithm (k=4)

Fig. 9 shows the minimum and maximum heights for the Batch Balanced Algorithm for k = 4. Both the minimum and maximum height have similar output.

5.1.4 Key Storage

Table 1 shows the minimum and maximum key storage for the Batch Balanced Algorithm.

Table 1
Minimum and Maximum Key Storage for
Batch Join and/or Depart Events

	Batch balanced algorithm
Min key storage	$\lfloor \log_k(N+J-D) \rfloor$
Max key storage	$\lfloor \log_k(N+J-D) \rfloor$

6 DISCUSSION

6.1 Revised batch rekeying algorithm

From the above simulations we observed that the algorithm works well only when J >= D. Our proposed revised variable length batch rekeying algorithm will try to avoid this condition to occur, so automatically leads to the minimum rekeying cost, update cost. But, the little overhead occur due to the batch rekey interval calculation. However this drawback can be shadowed due to its overall efficiency.

6.1 Optimization

From the above simulations, we observe that the Batch Balanced Algorithm has identical rekeying costs compared to existing algorithms when the number of joining members and the number of departing members are comparable. Therefore, one optimization that we can apply to our Batch Balanced Algorithm is not to reorganize the members in the key tree for the following condition:

$$D \leq J \leq (D - D_{min}) + kD_{min} \quad (3)$$

where D_{min} is the number of departing members at the minimum height.

For the case where J is equal to D, we replace all D departs by J joins. If J is greater than D and provided that J is smaller or equal to [(D-Dmin)+kDmin], then we replace all [D-Dmin] departs at the maximum height with [D-Dmin] joins. The remaining joining members are split across the D_{min} nodes.

Fig. 10 shows the update messages for our revised Batch Balanced Algorithm for k = 2. We can see that there are some cases where no update message is needed since there is no reorganization in the group. But, The rekeying costs still remain the same.

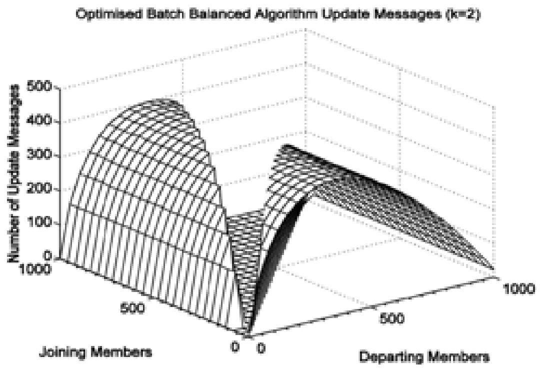


Fig. 10. Update message for the revised Batch Balanced Algorithm(k=2).

There is no way to maintain a balanced key tree without reorganizing the key tree when the number of departing members is greater than the number of joining members. So , it is necessary to avoid the condition J<D as much as possible.

By reducing the number of update messages we can reduce rekeying cost. This will achieved only by avoiding the condition J<D as much as possible. This can be achieved by our proposed work.

6 CONCLUSION

In this paper, we have presented revised variable length batch rekeying algorithm along with batch balanced algorithm. This algorithm tries to minimize the difference in height in the key tree without adding extra network costs. However, the algorithms require the GC to update the affected members on their node position by using update messages also it need to calculate its rekey interval. By minimizing the differences in height, we minimize the number of key storage and decryptions needed by each member. This is critical for terminals with limited computation and storage. Furthermore, reducing the number of decryptions can help to reduce the energy consumption, which, in turn, leads to battery saving. For batch join events, the way the joining members are inserted has a significant effect on the key tree, especially when there are a large number of join requests in a batch. The key tree can become unbalanced even if the insertion is at the minimum height. Existing algorithms do not simultaneously consider both the balancing of key tree and rekeying costs and therefore lead to either an unbalanced key tree or high rekeying costs. Our proposed Algorithm provide a good compromise compared to existing algorithms, producing a balanced key tree with low rekeying costs.. As for other events, our Batch Balanced Algorithm outperforms existing algorithms when the number of joining members is greater than the number of departing members and when the number of departing members is around N=k with no joining member. However, our algorithm try to avoid the condition J < D as much as possible and provide optimal solution in terms of rekeying cost, update messages. We further observe that, if we are able to group the members according transmission error rate than in conventional environments [19]. to their departing probability, then we are able to predict the rekeying costs based on the number of joining members. However, if the departing members are spread evenly across the key tree, then the highest rekeying cost happens at around N=k since most or all of the KEKs that the members store cannot be used.

REFERENCES

[1] Wee Hock Desmond Ng, Michael Howarth, Zhili Sun, and Haitham Cruickshank, "Dynamic Balanced Key Tree Management for Secure Multicast Communications," *IEEE Trans. Computers*, vol.56, no. 5, pp.590-605, May 2007.

[2] C. Wong, M. Gouda, and S. Lam, "Secure Group Communication Using Key Graphs," *IEEE/ACM Trans. Networking*, vol. 8, pp. 12- 23, Feb. 2000.

[3] D. R. Stinson and T. van Trung, "Some new results on key distribution patterns and broadcast encryption," *Designs, Codes and Cryptography*, vol. 14, no. 3, pp. 261–279, 1998.

[4] S. Paul, Multicast on the Internet and Its Applications. Kluwer Academic, 1998.

[5] U. Varshney, "Multicast over Wireless Networks," *Comm. ACM*, vol. 45, no. 12, pp. 31-37, Dec. 2002.

[6] U. Varshney, "Multicast Support in Mobile Commerce Application," *Computer,* vol. 35, no. 2, pp. 115-117, Feb. 2002.

[7] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Noar, and B. Pinkas, "Multicast Security: A Taxonomy and Efficient Constructions," *Proc. IEEE INFOCOM,* vol. 2, pp. 708-716, Mar. 1999.

[8] H. Harney and C. Muckerhirn, "Group Key Management Protocol (GKMP) Specification," *IETF RFC* 2093, July 1997.

[9] D.M. Wallner, E.J. Harder, and R.C. Agee, "Key Management for Multicast Issues and Architectures," *IETF RFC* 2627, June 1999.

[10] D. Balenson, D. McGrew, and A. Sherman, "Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization," Internet Draft, draft-irtf-smug-groupkeymgmt-oft- 00.txt, Aug. 2000.

[11] M. Valdvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The Versakey Frameworks: Versatile Group Key Management," *IEEE J. Selected Areas in Comm. (JSAC)*, vol. 17, no. 9, pp. 1614-1631, Sept. 1999.

[12] M.P. Howarth, S. Iyengar, Z. Sun, and H. Cruickshank, "Dynamics of Key Management in Secure Satellite Multicast," *IEEE J.Selected Areas in Comm. (JSAC)*, Feb. 2004.

[13] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting," *Proc. ACM SIGCOMM*, vol. 27, pp. 277-288, Sept. 1997.

[14] B.DeCleene et al., "Secure Group Communication for Wireless Networks," *Proc. Military Comm. Conf. (MILCOM)*, Oct. 2001.

[15] A. Perrig, "Efficient Collaborative Key Management Protocol for Secure Autonomous Group Communication," *Proc. Int'l Workshop CrypTEC*, 1999.

[16] X.S. Li, Y.R. Yang, M. Gouda, and S. Lam, "Batch Rekeying for Secure Group Communications," *Proc. 10th Int'l WWW Conf.*, May 2001.

[17] S. Setia, S. Koussih, and S. Jajodia, "Kronos: A Scalable Group Rekeying Approach for Secure Multicast," *Proc. IEEE Symp. Security and Privacy*, 2000.

[18] X.B. Zhang, S. Lam, D.Y. Lee, and Y.R. Yang, "Protocol Design for Scalable and Reliable Group Rekeying," *IEEE/ACM Trans. Networking*, vol. 11, pp. 908-922, Dec. 2003.

[19] Mingyan Li, R. Poovendran and C. Berenstein, "Design of Secure Multicast Key Management Schemes With Communication Budget Constraint" *IEEE Communications Letters*, vol. 6, no. 3, pp. 108-110, Mar. 2002

[20] J. Pegueroles and F. Rico-Novella, "Balanced Batch LKH: New Proposal, Implementation and Performance Evalution," *Proc.IEEE Symp. Computers and Comm. (ISCC)*, June 2003.

[21] P.P.C. Lee, J.C.S. Lui, and D.K.Y. Yau, "Distributed Collaborative Key Agreement Protocols for Dynamic Peer Groups," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* Nov. 2002.

[22] A.M. Eskicioglu, "Multimedia Security in Group Communication: Recent Progress in Key Management, Authentication and Watermarking,"*ACM Multimedia*

Systems J., special issues on multimedia security, pp. 239-248, Sept. 2003.

[23] W. Ng and Z. Sun, “Multi-Layers LKH,” *Proc. IEEE Int’l Conf. Comm. (ICC)*, May 2005.

[24] M.J. Moyer, J.R. Rao, and P. Rohatgi, “Maintaining Balanced Key Trees for Secure Multicast,” Internet Research Task Force (IRTF), Internet draft, draft-irtf-smug-key-tree-balance-00.txt, June 1999.

[25] W.H.D Ng, H. Cruickshank, and Z. Sun, “Scalable Balanced Batch Rekeying for Secure Group Communication,” *Elsevier Computers and Security*, vol. 25, pp. 265-273, June 2006.

Joe Prathap P M received the B.E. and M.E. degree in Computer Science & Engineering from Anna University in 2003 and 2005. He has registered for his Ph.D and has published papers in national and international conference, in the area of multicast security. He is currently a senior lecturer at Kalasalingam University. He is a member of ISTE, India.

Dr. V Vasudevan obtained his Ph.D degree from Madurai Kamaraj University, in 1991, He is working as a Senior Professor and Head in the Department of Information Technology. He has 25 years of teaching experience. He is guiding many research scholars and has published many papers in national and international conference and in many international journals. He has visited many universities in UK. He is a member of ISTE, India.