ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2, Special Issue 12, April 2016.

Robust Approach for Secure Multicast Communication using Batch Balanced Technique

J.Punitha Nicholine*

*Mrs. J.Punitha Nicholine, Assistant Professor, Department of Computer Science and Engg, PSNA college Of Engineering and Technology & Dindigul
Email Id:infantia.1@gmail.com

***Abstract*—**A secure multicast communication is important for applications such as pay-per-view 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. In this paper two Merging Algorithms suitable for batch join requests. To additionally handle batch depart requests, we extend these two algorithms to a Batch Balanced Algorithm. Simulation results show that our three algorithms not only maintain a balanced key tree, but their rekeying costs are lower compared with those of existing algorithms.

Keywords— Pay-per-view, group key management, secure group communication, rekeying.

1. INTRODUCTION

INTERNET Protocol (IP) multicast allows a sender to transmit a single copy of some data, with network elements such as routers making copies as necessary for the receivers. This approach reduces sender-processing overhead and network bandwidth usage.

Before these group-oriented multicast applications can be successfully deployed, access control mechanisms 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. 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.

The rekeying cost of the key tree approach increases with the logarithm of the group size for a join or depart request. 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 .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. In this scheme, the GC does not perform rekeying immediately; instead, it consolidates the total number of joining and departing members during a time period before performing the rekeying.

The efficiency of the key tree approach critically depends on whether the key tree is balanced . 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 outdegree 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

All Rights Reserved @ 2016 IJARMATE

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2, Special Issue 12, April 2016.

perform N - 1 decryptions to get the group key.

This paper propose two Merging Algorithms suitable for batch join events for combining subtrees together. These two Merging Algorithms not only balance the key tree, but have lower rekeying costs compared to existing algorithms. In other words, our Merging Algorithms allow all members in the multicast session to have similar storage and decryption requirements during each rekeying operation. Having a balanced key tree greatly benefits mobile devices since they generally have limited storage and computation power.

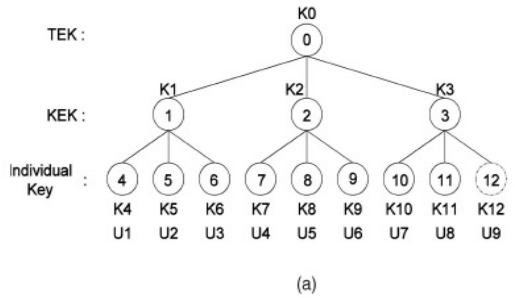

Fig1. Logical key tree

Reducing the number of decryptions needed by the mobile devices can help to conserve the battery power. In order to additionally handle departing members, we extend these two Merging Algorithms to a Batch Balanced Algorithm 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. Simulation results show that our Batch Balanced Algorithm performs significantly better than existing algorithms 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. For similar numbers of joining and departing members, our Batch Balanced Algorithm achieves the same performance as that of existing algorithms.

2. BACKGROUND

2.1 Key Tree Approach

In a typical key tree approach as shown in Fig.1, 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.

Fig. 1, 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. 1,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. 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. For example, suppose member U9 is departing in Fig. 1. Then, all the keys that it stores (K0 and K3) must be changed, except for its individual key. Let {x}y denote key x encrypted with key y and x0 denote the new version of key x. Then, the GC needs to multicast the rekey messages {K3'}K10, {K3'}K11, {K0'}K1, {K0'}K2, and {K0'}K3' to the members, giving a total of five encrypted keys.

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 logk N messages when the key tree is balanced. Suppose member U9 is joining the group. Then, the GC needs to multicast the following rekey messages to the members: {K3'}K3,{K3'}K12,{K0'}K0, and {K0'}K3'. 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 logk N. However, if the key tree becomes unbalanced, the distance from the root to a leaf node can become as high as N.

2.2 Batch Rekeying

Batch rekeying is when join/depart requests are collected during a time interval called the rekey interval and are rekeyed together It also reduces

All Rights Reserved @ 2016 IJARMATE

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2, Special Issue 12, April 2016.

the number of group rekey events. Furthermore, the number of rekey messages that need to be multicast to the group can be much smaller than the number of rekey messages that would be generated if each membership change were processed individually due to the overlapping in paths from the leaf nodes to the root.

2.3 Related Work

Some notations and definitions are used in this paper. The term ST to indicate a subtree. The "minimum height" is used to mean the minimum number of levels in a tree or subtree from the root to any leaf node. Similarly, the "maximum height" is used to mean the maximum number of levels in a tree or subtree from the root to any leaf node. The following are the variables that are defined in this paper:

D	Number of departing members
J	Number of joining members
h	Height of key tree $(1 + \log_k N)$
H_{MIN}	Minimum height of leaf node in key tree
H_{MAX}	Maximum height of leaf node in key tree
H_{INSERT}	H_{MIN} of ST_A - H_{MAX} of ST_B

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.

In Marking Algorithm1, 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. By the term "shallowest node," we mean the leaf node of minimum height in our terminology. 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.

In 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. Balanced Batch Logical Key Hierarchy (LKH),has also been proposed to alleviate the inefficiency in Marking Algorithm 1 but this algorithm is only suitable for a binary key tree (k=2) and the author does not offer a solution for a key tree with other outdegrees.

3. BATCH REKEYING ALGORITHM

This paper 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. Assume that ST_A has a greater height than ST_B and both subtrees are of the same outdegree 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. The criteria for choosing Merging Algorithm 1 are when the difference between HMAX ST A and HMIN ST B is greater than 1 and when the difference between HMAX ST A and HMAX ST B is greater than or equal to 1. If both of these conditions are fulfilled, then the algorithm calculates HINSERT . 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 HINSERT or level HINSERT-1. If HINSERT=0, then levels 0 and 1 are searched.

All Rights Reserved @ 2016 IJARMATE

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2, Special Issue 12, April 2016.

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 HINSERT for insertion as follows: If HINSERT =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 HMIN_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(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

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 HMAX_ST_A and both HMIN_ST_B and HMAX_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

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.

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. Do the steps up to 4.These usable subtrees ST1 to ST7 are identified as shown in Fig 2. Now follow the steps 5 and 6.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 or Marking Algorithm 2 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.

3.4 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

All Rights Reserved @ 2016 IJARMATE

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2, Special Issue 12, April 2016.

of their new location. 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 m0 by using the following function:

$$f(m0) = kx(m' - m) + m0;$$

where x denotes the level of the usable key tree.

Update Messages	Rekey Messages				
11, 23 for U9 and U10	{K31}K35	{K33}K39	{K41}K1	{K43}K45	{K45}K11
13, 24 for U13 and U14	{K31}K36	{K33}K40	{K41}K42	{K43}K46	{K45}K13
26, 29 for U12	{K32}K37	{K34}K26	{K42}K43	{K44}K33	{K46}K31
	{K32}K38	{K34}K30	{K42}K44	{K44}K34	{K46}K32

4. PERFORMANCE EVALUATION

This section shows the performance of our proposed algorithms and compare them with the Marking Algorithms. We consider four performance metrics: rekeying cost, update cost, minimum and maximum height in the key tree, 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.

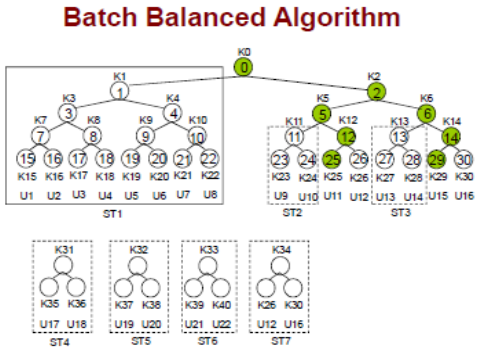

Fig.2

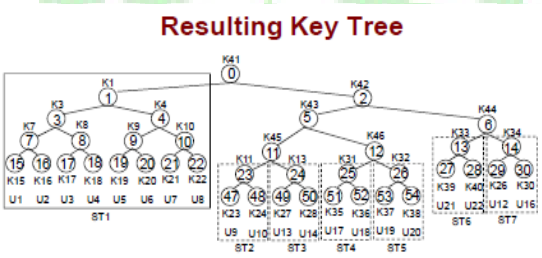

Fig.3

A higher rekeying cost means that more bandwidth is needed for the transmission. Although Marking Algorithm 2 adopts the User-Oriented Key Assignment Algorithm (UKA) where all of the encryptions for a member are assigned in a single packet, we ignore the UKA when we calculate the rekeying costs since it leads to a significant number of duplications in rekey messages. Instead, we just calculate the total number of rekey messages that are needed without any duplication. 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.

4.1 Merging Algorithm Performance Evaluation:

Some simulations to compare the performance of both of our Merging Algorithms with existing work.

4.1.1 Rekeying Cost:

In Fig4, we can see that Marking Algorithm 2 has the highest rekeying cost. This is because the joining members are inserted one by one at each leaf node, which affects the paths from the affected leaf nodes to the root. As the number of joining members increases, the number of affected nodes increases significantly. On the other hand, other three algorithms have similar rekeying costs since they try to minimize the number of affected nodes. Marking Algorithm 1 minimizes the rekeying costs by placing the new subtree, which consists of joining members and one removed member on the shallowest height, at the old location of the removed member. Merging Algorithm 1 inserts the new subtree consisting of the joining members into one of the key nodes in the key tree at a location that depends on the number of the joining members; thus, as the number of joining members increases, the number of affected nodes is reduced since the key node selected for insertion gets closer to the root. For Merging Algorithm 2, a new root is created with the existing subtree and the new subtree consisting of the joining members, which are inserted as its children.

4.1.2 Update Cost

Of the four algorithms, only Marking Algorithm2 does not need to distribute update messages to the members. Marking Algorithm 1 needs to send one update message to inform the removed leaf node of its new location. Similarly, both Merging Algorithms need to send out one update message to

All Rights Reserved @ 2016 IJARMATE

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2, Special Issue 12, April 2016.

inform the affected members of the newly created node.

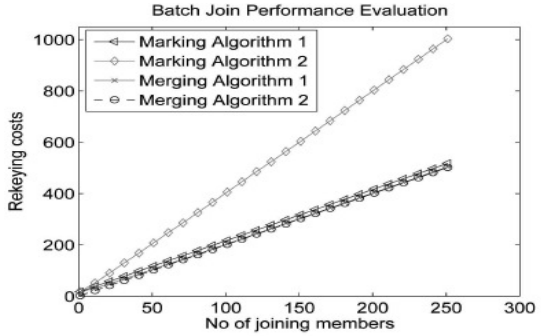

Fig. 4 Batch join rekeying costs

4.1.3 Minimum and Maximum Height

Fig 5 shows the maximum height of the key tree after the joining members have been inserted into the key tree for all algorithms. Only Marking Algorithm 2 and Merging Algorithm 2 maintain at a fixed height, regardless of the number of joining members. Marking Algorithm 2 alleviates the inefficiency of Marking Algorithm 1 by inserting the joining members one by one at each leaf node, whereas Merging Algorithm 2 creates a new root and inserts the existing key tree and the joining member key tree as its children. Merging Algorithm 1 has the same performance as Marking Algorithm 2 and Merging Algorithm 2 when the number of joining members is less than or equal to half the group size. However, once the number of joining members exceeds half the group size, the maximum height increases by 1. Fig 6. shows the maximum difference in height of the key tree, which indicates whether the key tree is balanced. The maximum difference in height for Marking Algorithm 1 increases considerably as the number of joining members increases. Similarly, our Merging Algorithm 2 is not a balanced key tree when the number of joining members is less than half the group size and it only maintains a balanced key tree when the number of joining members is greater than or equal to half the group size. As for our Merging Algorithm 1, it maintains a balanced key tree when the number of joining members is less than or equal to half the group size. The difference in height in Merging Algorithm 1 increases by 1 once the number of joining members exceeds half the group size since the child of the root is selected for the insertion. Marking Algorithm 2 is the only algorithm that creates a balanced key tree, regardless of the number of joining members. However, this comes with the drawback of the high rekeying costs.

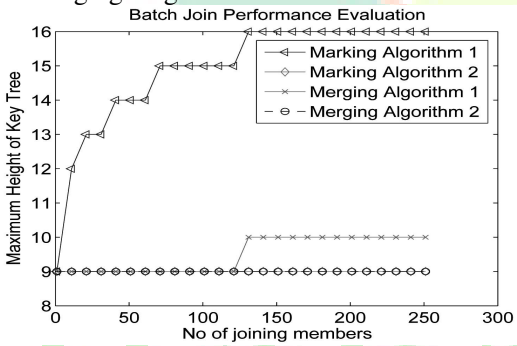

Fig 5.Maximum height of the key tree

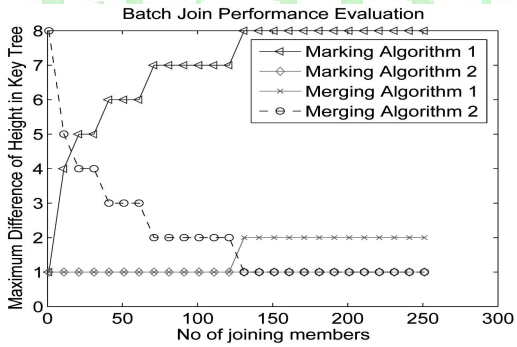

Fig 6.Maximum difference in height

4.1.4Key Storage:

It shows the minimum and maximum number of keys that a member needs to store for the four algorithms for batch join events. the maximum number of keys that a joining member needs to store in Marking Algorithm 1 is dependent on the number of joining members at that particular interval. A large number of joining members results in a great difference in key storage among members. Marking Algorithm 2 does not suffer from the storage inefficiency as in Marking Algorithm 1, but it comes at the expense of the large rekeying costs.

5. FUTURE WORK:

To implement 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 . Keeping this point in mind, we can apply variable batch rekey interval for rekeying. It minimizes the number of key update messages. In order to reduce the computation we are going to separate the mobile and non mobile nodes in a tree and if there is scalable tree then centralized server is

All Rights Reserved @ 2016 IJARMATE

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2, Special Issue 12, April 2016.

assigned in the mobile node side to perform the operations.

REFERENCES:

[1] S.E. Deering, "Host Extensions for IP Multicasting," IETF RFC 1112, Aug. 1989.
[2] S. Paul, *Multicast on the Internet and Its Applications*. Kluwer Academic, 1998.
[3] U. Varshney, "Multicast over Wireless Networks," *Comm*. ACM, vol. 45, no. 12, pp. 31-37, Dec. 2002.
[4] U. Varshney, "Multicast Support in Mobile Commerce Application," *Computer*, vol. 35, no. 2, pp. 115-117, Feb. 2002.
[5] H. Harney and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture," IETF RFC 2094, July 1997.
[6] 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.
[7] A. Ballardie, "Scalable Multicast Key Distribution," IETF RFC 1949, 1996.
[8] 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.
[9] 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.
[10] H. Harney and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture," IETF RFC 2094, July 1997.

IJARMATE
Your ultı-MATE Research Paper !!!

All Rights Reserved @ 2016 IJARMATE