Key Management Using K-Dimensional Trees

Renuka A [#1], K.C. Shet [*2]

[#]*Department of Computer Science And Engg. MIT, Manipal,Karnataka-576104, India*

[1] renuka.prabhu@manipal.edu

[*] *Department of Computer Engg., NITK, Surathkal, Karnataka-575025, India*

[2] kcshet@rediffmail.com

***Abstract*— In this paper we present a protocol for group key management in mobile ad hoc networks based on K-dimensional trees, a space partitioning data structure. We use a 2-dimensional tree for a 2 dimensional space. The 2 dimensional tree resembles a binary tree. The protocol reduces the memory requirements for storing the tree by nearly 50% compared to the existing methods and also reduces the number of key changes required whenever membership changes occur.**

I. INTRODUCTION

An ad hoc network is a collection of autonomous nodes that communicate with each other, most frequently using a multi-hop wireless network. Nodes do not necessarily know each other and come together to form an ad hoc group for some specific purpose. Key distribution systems usually require a trusted third party that acts as a mediator between nodes of the network. In key agreement protocols [1] nodes interact with each other to compute a common key. Key agreement protocols do not usually require a trusted authority. A number of such protocols [2], [3], [4] use Diffie-Hellman style key agreement approach for groups. A shortcoming of these protocols for ad hoc networks is that they usually require a broadcast channel and hence routing infrastructure. Ad hoc networks typically do not have an online trusted authority but there may be an off line one that is used during system initialization. A node in an ad hoc network has direct connection with a set of nodes, called neighbouring nodes, which are in its communication range. The number of nodes in the network is not necessarily fixed. New nodes may join the network while existing ones may be compromised or become un-functional. Moreover, a leave or join would normally require the whole protocol of key management to start over again.

Many secure group communication systems rely on a group key, which is a secret shared among the members of the group. Secure messages are sent to the group by encrypting them with the group key. Because group membership is dynamic, it becomes necessary to change the group key in an efficient and secure fashion when members join or leave the group. The group may require that membership changes cause the group key to be refreshed. Changing the group key prevents a new member from decoding messages exchanged before it joined the group. If a new key is distributed to the group when a new member joins, the new member cannot decipher previous messages even if it has recorded earlier messages encrypted with the old key. Additionally, changing the group key prevents a leaving or expelled group member from accessing the group communication (if it keeps receiving the messages). If the key is changed as soon as a member leaves, that member will not be able to decipher group messages encrypted with the new key. However, distributing the group key to valid members is a complex problem. Although rekeying a group before the join of a new member is trivial (send the new group key to the old group members encrypted with the old group key), rekeying the group after a member leaves is far more complicated. The old key cannot be used to distribute a new one, because the leaving member knows the old key. Therefore, a group key distributor must provide another scalable mechanism to rekey the group. We present a protocol for solving this problem. The protocol attempts to minimize the worst case communication cost of updating the group key. We focus on the trade-off between the communication cost of updating the key and the number of hops in communication.

The rest of the paper is organized as follows. Section II focuses on the related work in this field. The proposed scheme is presented in Section III. Performance analysis of protocol and conclusion are given in Section IV and Section V respectively.

II. RELATED WORK

Securing communication in ad hoc networks requires key management systems that provide support for dynamic properties. The different approaches to group key management are divided into three main classes:

—*Centralized group key management protocols.*
A single entity is employed for controlling the whole group, hence a group key management protocol seeks to minimize storage requirements, computational power on both client and server sides, and bandwidth utilization.

—*Decentralized architectures.* The management of a large group is divided among subgroup managers, trying to minimize the problem of concentrating the work in a single place.

—*Distributed key management protocols.*
There is no explicit Key Distribution Controller (KDC), and the members themselves do the key generation. All members can perform access control and the generation of the key can be either contributory, meaning that all members contribute some information to generate the group key, or done by one of the members.

There are four important security properties of group key agreement that need to be satisfied.

(1) *Group Key Secrecy* guarantees that it is computationally infeasible for a passive adversary to discover any group key.

(2) *Forward Secrecy* guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover any subsequent group key.
(3) *Backward Secrecy* guarantees that a passive adversary who knows a contiguous subset of new group keys cannot discover preceding group key.
(4) *Key Independence* guarantees that a passive adversary who knows a proper subset of group keys cannot discover any other group key.

A. Logical Key hierarchy

Wong et al. [5] proposed the use of a Logical Key Hierarchy (LKH). In this approach, a Key Distribution Controller (KDC) maintains a tree of keys. The nodes of the tree hold key encryption keys. The leaves of the tree correspond to group members and each leaf holds a key encryption key (KEK) associated with that one member. Each member receives from the KDC, a copy of the KEK associated with its leaf and the KEK's corresponding to each node in the path from its parent leaf to the root. The key held by the root of the tree is the group key. For a balanced tree, each member stores at most $\log_2 n + 1$ keys, where $\log_2 n$ is the height of the tree.

The algorithm proposed by Waldvogel et al. [6] is different for joining operations. Instead of generating fresh keys and sending them to members already present in the group, all keys affected by the membership change are passed through a one way function. Every member that already knew the old key can calculate the new one. Hence, the new keys do not need to be sent and every member can calculate them locally. However, this information has to be sent to the members by the KDC. This algorithm is known as LKH+.

B. One-way Function Tree

An improvement in the hierarchical binary tree approach is a one-way function tree (OFT) and was proposed by McGrew and Sherman [7]. Their scheme reduces the size of the rekeying message from $2(\log_2 n)$ to only log2n. Here a node's Key Encryption Key (KEK) is generated rather than just attributed. The KEK's held by a node's children are blinded using a one-way function and then mixed together using a mixing function. The result of this mixing function is the KEK held by the node

C. Distributed Logical Key Hierarchy

A distributed approach based on the logical key hierarchy is suggested by Rodeh et al. [8]. In this approach, the Key Distribution Controller or the Group Controller (GC) is completely abolished and the logical key hierarchy is generated among the members, therefore there is no entity that knows all the keys at the same time. This protocol uses the notion of sub trees agreeing on a mutual key. This means that two groups of members, namely sub tree L and sub tree R, agree on a mutual encryption key. This algorithm takes $\log_2 n$ rounds to complete, with each member storing $\log_2 n$ keys.

In a large and highly dynamic group, to reduce the rekeying overhead, the GKC may choose to rekey in batches as in [9]. In batch rekeying the departing members continue to get access to the group data for a brief period after they leave so that forward secrecy fails. The joining members are put on hold until the next rekeying instance.

In Enhanced One Way Function Tree (EOFT) [10],[11], the Group Key Controller (GKC) constructs a rooted balanced tree. Every group member is associated with a unique leaf node. The root represents the group key. Each internal node represents a logical subgroup. Each member shares a secret key with the GKC through the registration protocol in a secured way by unicast. Each member needs the secret keys of the internal nodes in its path to root for computing the group key. But the GKC does not send all these internal node keys to the members directly. Instead it sends only $\log_2 n$ pseudo key tuples of the siblings of those internal nodes that are needed to compute their ancestors' keys.

III. PROPOSED SCHEME

A. K- DIMENSIONAL TREE

A K-Dimensional tree (KD tree) [12] is a space-partitioning data structure for organizing points in a K-Dimensional space. A KD tree uses only splitting planes that are perpendicular to one of the coordinate system axes. Since there are many possible ways to choose axis-aligned splitting planes, there are many different ways to construct KD trees. The canonical method of KD tree construction has the following constraints: As one moves down the tree, one cycles through the axes used to select the splitting planes. (For example, the root would have an x-aligned plane, the root's children would both have y-aligned planes and the root's grandchildren would all have z-aligned planes, and so on.) At each step, the point selected to create the splitting plane is the median of the points being put into the KD tree, with respect to their coordinates in the axis being used. (Note the assumption that we feed the entire set of points into the algorithm.) This method leads to a balanced KD tree, in which each leaf node is about the same distance from the root. For the location of the points (indicated by dots) in Fig.1 (a) the KD tree obtained is as shown in Fig. 1(b)

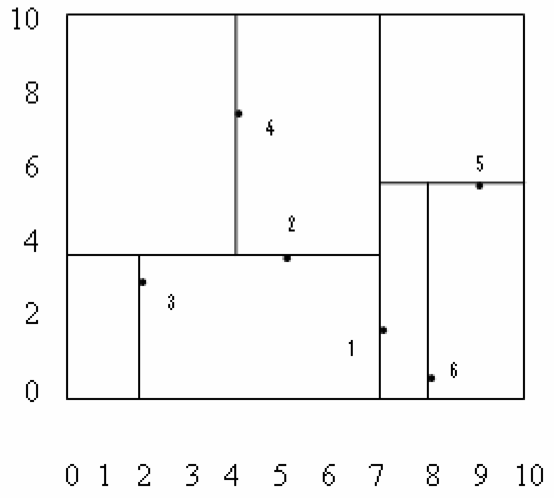

Fig. 1(a). 2-D grid

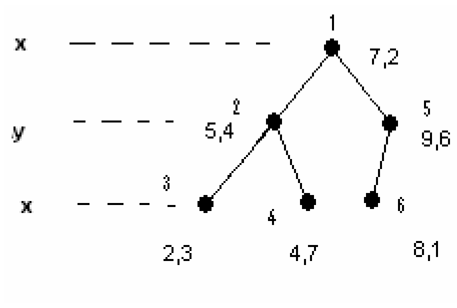

Fig. 1(b). 2-D tree for the grid shown in Fig.1 (a)

The KD tree model consists of the root node and child nodes as shown in Fig.2. It resembles a binary tree.

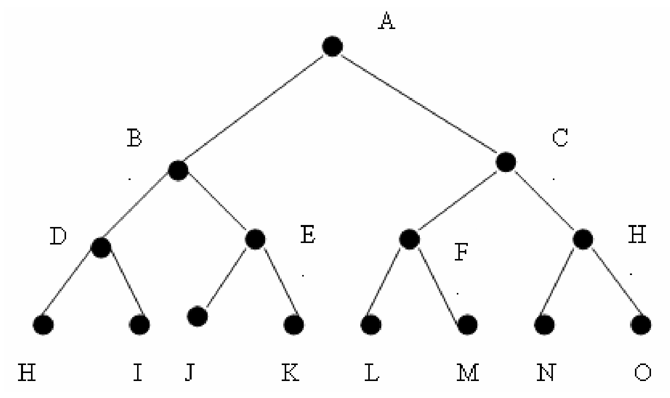

Fig. 2. A sample 2-D tree

B. KD Tree Model

In this paper, we propose a practically secure protocol to build efficient K Dimensional tree model for mobile ad hoc networks which have no trusted third party to maintain the keys and uses the deployment knowledge such as location of the sensors for the construction of the trees. We assume that the approximate location of the sensor nodes is known and is loaded into the sensor nodes before deployment. Our proposal employs a secure distributed model based on secret-key encryption algorithm and one-way hash function. The protocol also includes group keying and group membership change operations (single join and leave). In this protocol, each member holds 3 keys when the tree is balanced. Group multicast message size for each addition and leave operation depends on the size of the subgroup and is independent of the number of members. Every node of a KD tree, from the root to the leaves, stores 3 keys. We assume that authentication and individual key exchange are provided, and focus on the secure management and distribution of the group and auxiliary keys to provide continued security in the face of membership change or group change. There are two types of security that a key distribution algorithm can provide: backward security, meaning that a new member is not able to decrypt past group communications, and forward security, meaning that a departing member is not able to decrypt future group communications. We define a key distribution algorithm to be secure if for any set of adversaries, after any sequence of update operations, the adversaries cannot obtain the group key, This definition provides both backward and forward security and is met if all keys are pseudo-random, all keys that member u receives as a result of an ADD (u) operation are new, and all keys that member u held before a DELETE (u) operation are removed.

A KD tree is a tree where each node is labelled with a key. If i is a node in a KD tree then the label of i is a group key for the subgroup G_i consisting of all the nodes of the sub tree rooted at i. The root label is the group key for the entire multicast group. The root is located at level 0 and the lowest leaves are at level *h*. Since a KD tree resembles a binary tree, every node is either a leaf or a parent of two nodes. The nodes are denoted by letters A, B, ….. Each node is associated with its private key K_i. Generally, a new key k is distributed to subgroup G_i using the key K_i (the label of node i) using the encrypted message E (k, K_i), where the key k is encrypted with the key K_i . Define communication cost as the number of encrypted messages required to update the keys due to group membership change. This is the metric we are interested in minimizing.

C. Subgroup formation and subgroup key agreement

Consider the KD tree shown in Fig.2 which represents an ad hoc group. It is a balanced tree consisting of 15 nodes and is of height 3. All members carry a preassigned session key which is used to encrypt the group keys generated. (How this is done is out of the scope of this paper). These 15 nodes are divided into 7 subgroups labelled G1, G2, G3,……G7 where each subgroup consists of 3 members as shown in Fig. 3. These 15 members can also be divided into 3 subgroups, each subgroup consisting of 7 members each, as shown in Fig.4. In all our discussions, we will consider 7 subgroups, each of 3 members. Each subgroup communicates with its members using the subgroup key generated. Each member contributes its share to the generation of a subgroup key. The keys held by a node's siblings are blinded using a one way function and then transmitted to the node. The node's key in turn is mixed using a mixing function. The result of this mixing function is the subgroup key for the subgroup headed by the node.

i.e. $K_{G4} = f\,(g(K_D\,), g(K_H\,), g(K_I\,))$

where g ( ) is a one way function and f ( ) is a mixing function

Thus, in Fig 2, the subgroup key K_{G4} for the subgroup G_4 is generated from K_H, K_I and K_D.

K_{G2} is generated by the knowledge of K_D K_B, and K_E. This key is the subgroup key for the subgroup K_{G2} and so on. Since location information is used in the construction of the tree, these members are also physical neighbours. If the node

H has to communicate to node O then, the messages have to be routed through D and then B which is received by C and then broadcasted in its subgroup which is then received by member G and retransmitted to O. Thus communication takes place in five hops.

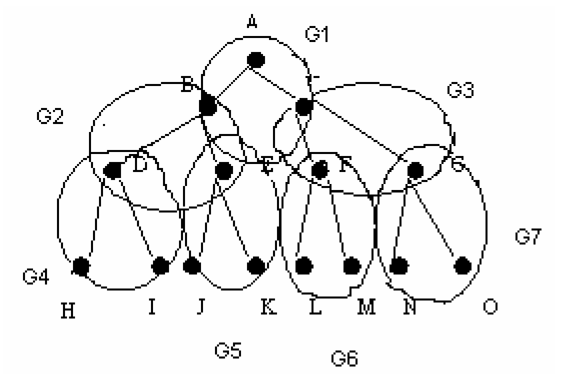

Fig. 3. 2-D Tree with 7 subgroups

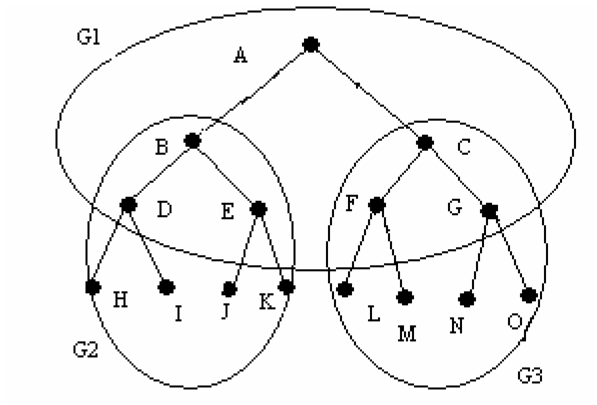

Fig. 4. 2-D Tree with 3 subgroups

D. Membership Changes

In this section, we will discuss the key agreement protocol when a new member joins or existing member leaves.

1) Member joins

When a new member joins, the group key has to be updated in the subgroup to which the new member joined, in order to prevent the new user from accessing the information previously exchanged. This is called rekeying. Suppose a new member joins, the insertion point in the tree is found by traversing the tree starting from the root and moving to either the left or the right child, depending on whether the point to be inserted is on the "left" or "right" side of the splitting plane. Once a leaf node is reached, the new point is added as either the left or right child of the leaf node, again depending on which side of the node's splitting plane contains the new point. This new node initiates communication with the parent node to which it is attached and sends the blinded version of its key. Suppose a new member P, gets attached to the node H, in Fig.3. Node H receives the blinded version K_P of node P and mixes with its own blinded K_H to get the new subgroup key. This new subgroup key is encrypted using the old subgroup key and multicast to all nodes except the new one. The same is unicast to the new member by its parent. Thus, only the members in the subgroup are involved in the generation of the new subgroup key. As the number of nodes increases beyond a threshold (3 in this case), the groups can be split and new group keys can be generated using the same technique

2) Member leaves

There are three cases.
Case (i): member at a leaf node leaves
Case (ii): member at an intermediate level leaves
Case (iii): member at the root node leaves

Case (i): When a member at a leaf node leaves, the members belonging to the subgroup to which the leaving member is attached, has to change the sub group key which is known to the leaving member. After member H leaves the updated tree appears as shown in Fig.5. Only the subgroup key K_{G4} needs to be changed. This is done by changing the mixing function used to get the subgroup key and is sent to the other siblings in a secure manner.

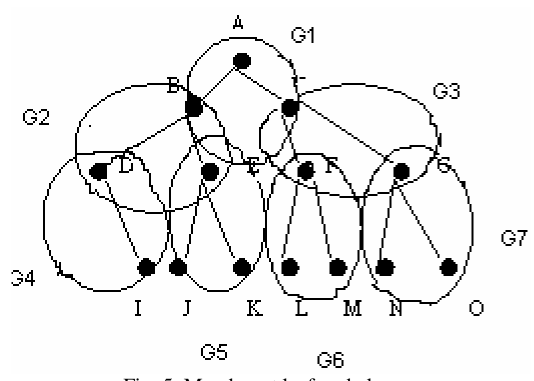

Fig. 5. Member at leaf node leaves

Case (ii): If a member is at the intermediate level (other than the root or the leaf), then the subgroup key belonging to two subgroups have to be changed. These subgroups are:
(i) The subgroup headed by the leaving node and
(ii) The subgroup in which the leaving node was the sibling
For example in Fig.3 let us consider that node B is leaving The subgroup keys K_{G1}, K_{G2} used by the subgroup G_1 and G_2 have to be changed. This is done by promoting the node D to the place of B, restructuring the tree and then generating the subgroup key in the similar manner as described above. The tree after the member leaves is as shown in Fig.6. This involves changing two subgroup keys and includes the contribution of 6 nodes.

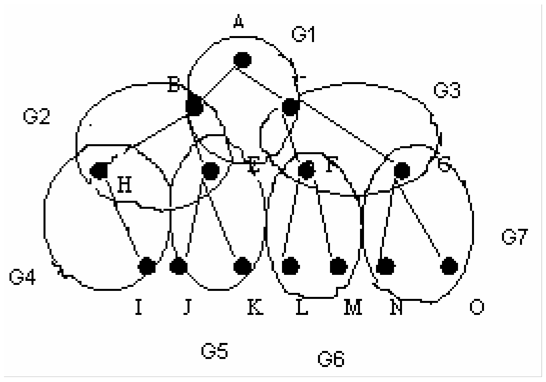

Fig. 6. Member at intermediate node leaves

Case (iii): When the member present at the root leaves, this case is similar to the previous case. Here, the node B gets promoted to the root position and requires the subgroup keys K_{G1} and K_{G2} of the subgroups G_1 and G_2 to be changed. The other subgroup keys remain the same.The updated tree appears as shown in Fig.7.

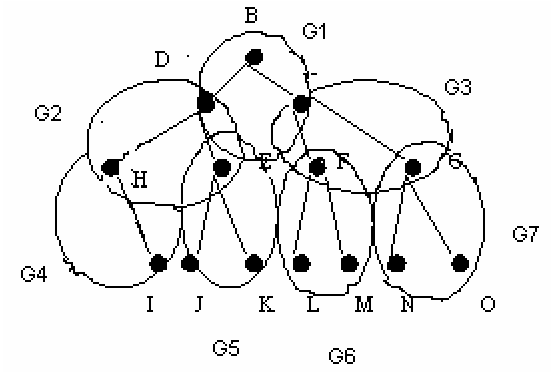

Fig. 7. Member at root node leaves

IV. Protocol Analysis

In our protocol, considering a subgroup of 3 members, each member needs to store 3 keys: its own secret key and 2 subgroup keys to which it is attached. Comparing this with the hierarchical binary tree (HBT) model, used in centralized Logical Key Hierarchy (LKH) and distributed Logical Key Hierarchy protocols of key management, each member needs to store all the keys on the path to the root which means $\log_2 n + 1$ keys. If we consider a tree of height 3, HBT model represents 8 members since only leaf nodes represent members and each member stores 3 keys. In our protocol, the same tree represents 15 members, each member storing 3 keys. The complexity of the tree structure is greatly reduced which in turn reduces the amount of memory required to store the tree by nearly 50%.

Whenever a new member joins or leaves the group, the tree has to be restructured and the corresponding subgroup keys have to be changed. The siblings communicate the blinded version of their key to the parent and the parent computes the subgroup key which is then communicated in a secure manner to the siblings. This requires one round of message transmission whereas in distributed LKH, it requires 3 rounds. Also the number of key changes in our protocol is at most 2 whereas in both centralized and distributed LKH. It needs $\log_2 n$ key changes whenever membership changes occur; where n is the total members in the group. This remains constant; irrespective of the number of members in the network as long as we consider that the subgroup consists of 3 members. In our model, it requires the transmission of rekeying messages to 4 members, when a subgroup of 3 members is considered. The graph of number of rekeying messages versus the total number of members for different number of subgroup members is shown in Fig.8. Here m denotes the subgroup size. The graph depicting the number of key changes for our protocol and HBT model is shown in Fig.9.

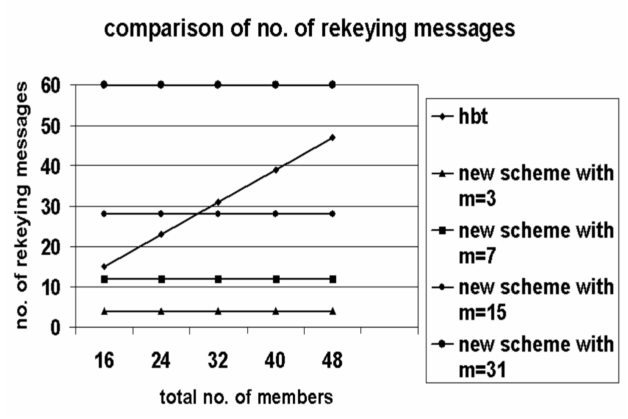

Fig. 8. Graph showing the variation of rekeying messages with respect to the total members for various subgroup sizes

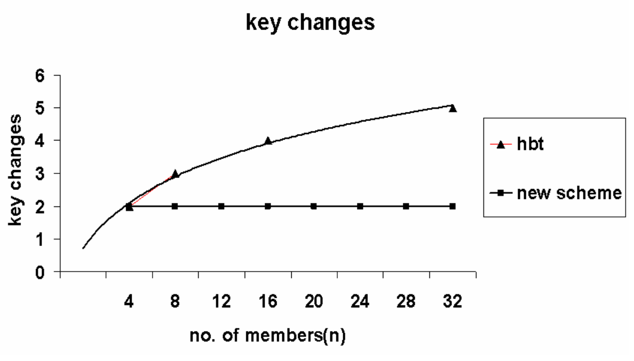

Fig. 9. Graph of number of key changes vs. number of members

V. Conclusion

We proposed a protocol for group key agreement that uses the location information. We have compared the performance of this protocol with the most commonly used hierarchical binary tree (HBT) model.

Suppose, we consider a system with 8 members, HBT model requires 3 group key changes and transmission of rekeying messages to 7 members. In our protocol, it requires 2 group key changes and transmission of rekeying messages to 4 members when a subgroup of 3 members is considered. This remains constant, irrespective of the number of members in the network; as long as we consider that the subgroup consists of 3 members. These values increase in HBT, as the number of members increases. In a network consisting of large number of nodes the subgroup can be comprised of seven members, 15 members and so on. As the number of members in the subgroup increases, the number of subgroups for a given network decreases, which in turn reduces the number of hops in communication. A trade off between the number of members in the subgroup and the hops in communication have to be done so that an optimal value for the subgroup chosen based on the requirements.

References

[1] W. Diffie and M. E. Hellman. "*New directions in cryptography*", IEEE Transactions on Information Theory, IT-22(6):644–654, 1976.

[2] M. Burmester and Y. Desmedt. "*Efficient and secure conference-key distribution*", In Proceedings of the International Workshop on Security Protocols, pages 119–129, London, UK, 1997. Springer-Verlag.

[3] E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. "*Provably authenticated group Diffie-Hellman key exchange*", In Proc. of ACM-CCS 01, page 255C264, Philadelphia, Pennsylvania, USA, November 2001. ACM, ACM Press.

[4] H.-J. Kim, S.-M. Lee, and D. H. Lee. "*Constant-round authenticated group key exchange for dynamic groups*." In Asiacrypt 2004, 2004.

[5] Wong, C. K., Gouda, M. G., and Lam, S. S. 2000. "Secure group communications using key graphs" *IEEE/ACM Trans. Netw. 8*, 1 (Feb.), 16–30.

[6] Waldvogel, M., Caronni, G., Sun, D., Weiler, N.,Aand Plattner, B. 1999. "The VersaKey framework: Versatile group key management." *IEEE J. Sel. Areas Commun. (Special Issue on Middleware) 17*, 9 (Aug.), 1614–1631.

[7] A.T. Sherman and D.A. McGrew, "*Key Establishment in Large Dynamic Groups Using One Way Function Trees*," IEEE Software Trans Engg., vol. 29, no. 5, pp. 444-458, May 2003

[8] Rodeh O., Birman, K.,and Dolev, D. 2000, "Optimized group rekey for group communication systems", In Network and Distributed System Security.(San Diego, Calif., Feb.).

[9] Heydari M. H, Morales L. And Sudborough I. H, "*Efficient Algorithms for Batch Re-keying Operations in Secure Multicast*", HICSS 2006, 39th Annual Hawaii International Conference on System Sciences, January 2006, pp. 218.

[10] Parvatha Varthini, S. Valli, "*Generation of Group key Using Enhanced One Way Function Tree Group Rekey Protocol*" Proceedings of the International Conference on Computing: Theory and Applications (ICCTA'07)

[11] Parvatha Varthini. B., Valli. S, "*EOFT: An Enhanced one way Function Tree rekey Protocol based on Chinese Remainder Theorem*", ISCIS05, Lecture Notes on Computer Science, Vol. 3733, pp 33-44.

[12] http://en.wikipedia.org/wiki/Kd-tree