Group Key Management Scheme for Simultaneous Multiple Groups with Overlapped Membership

Purushothama B R
Computer Science and Engineering
National Institute of Technology
Warangal, INDIA
Email: puru@nitw.ac.in

B B Amberker
Computer Science and Engineering
National Institute of Technology
Warangal, INDIA
Email: bba@nitw.ac.in

***Abstract*—Consider several groups engaged in a single project. Group members of one group happens to communicate with other group members. How might the communication among the members be carried out securely? How to ensure that a group member of one group communicate securely with its own group and with other group members as well?. We predict that secure group key management schemes for simultaneous multiple groups which allow inter-group communication become increasingly popular. We detail this problem under secure group communication model. Following the work of [1], we provide an efficient secure multiple groups key management scheme with overlapping membership based on the proposed *Key-User Tree* structure with following properties. $a)$ Scheme handles multiple groups simultaneously. $b)$ Group members within a group can communicate securely. $c)$ Group members of one group can communicate with other group members securely. We analyze join and leave protocols of the scheme for storage, encryption and key changes upon membership change. Our scheme scales well as the overlapping memberships across the multiple groups increases rapidly. Our scheme achieves significant reduction in rekeying cost, storage and scales well in comparison with [1].**

I. Introduction

A group is a set of users. The users of the group are called group members. A group can comprise of users playing an online game with each other. Group can consist of participants in a multimedia conference. Likewise, there are several instances and applications which inherently require the participation of users as a group.

Group communication is a means of facilitating the members of the group to exchange messages with each other. The users not part of the group are termed as *non-members* of the group. In group communication, the messages exchanged among the group members should be protected from access by *non-members* of the group. The exchanged messages within the group should be secure.

Secure group communication deals with the aspect of protecting the group messages from the *non-members* of the group. Encryption can be used to provide the confidentiality of the communication in the group. The messages are encrypted using the *chosen key*, which is termed as the *group key* in group communication context. Only those users who possess *group key* can recover the encrypted message. If encryption employed to protect the messages of the group, distributing the cryptographic keys to members of the group securely becomes a challenging issue. This coins the problem of *secure group key management*.

Generally, group can be categorized into two based on the **join/leave** ((to group) / (from group)) activities of the users.

1) **Static group**: In the *static group*, existing group members never *leave* and new members never *join* the group for the entire lifetime of the group.
2) **Dynamic group**: In *dynamic group*, new members join the group and existing members leave the group during the lifetime of the group.

Secure group communication in dynamic group is challenging in comparison to static group. The *group key* used to protect the messages in the static group need not be changed for the entire lifetime of the group. Since there are no member leaves and joins in the group, the same group key can be used throughout. In dynamic group, the group member leaves the group and becomes *non-member*. The *group key* of the group after the group member leaves should be changed. Otherwise, the user who left the group can read the future communications of the group being currently a *non-member* of the group. This violates the basic idea of *secure group communication.* So, the group key is changed to disallow the user who left the group from accessing the future communications of the group. In the similar line, when the new user joins the group, it should not be given the same current *group key*. If given the same *group key*, he could decrypt the past messages that he might have collected in the past when it was a *non-member* of the group. So, the *group key* is changed after a new member joins the group.

To summarize, the *group key* must be changed for every membership changes. This introduces need of *access control* in *secure group communication*. Based on the discussion above, there are two categories of access controls that should be provided.

1) **Forward access control**: It is used to prevent leaving or expelled group member (member can voluntarily leave the group or compelled to leave the group on detection

978-1-4244-8953-4/11/$26.00 © 2011 IEEE

of malicious behavior) from accessing the future group communications.

2) **Backward access control**: It is used to prevent a new member joining the group from reading the messages that were exchanged before. In other words, prevent the newly joined member from reading the past communications.

The process of changing the *group keys* to provide *forward access control* and *backward access control* is called *rekeying*.

As it is evident from above discussion, *group key management* plays an important role in enforcing the access control on the *group key*. There are several different approaches to group management proposed by the literature. The approaches are categorized into following,

- **Centralized group key management schemes or protocols**: These schemes employ a trusted centralized entity called *Key Distribution Center (KDC)* which controls the whole group [2][3].
- **Decentralized group key management schemes or protocols**: In these schemes the responsibility of managing the large group is divided among subgroup managers[4].
- **Distributed or contributory group key management schemes and protocols**: There is no *KDC* in this scheme and all group members contribute in generating the group key [5] [6]. Access control is also done by the group members.

The classical survey to the area of *group key management schemes* that fall into the three categories elaborated above is given in [7].

One of the efficient, scalable centralized group key management scheme is provided by Wong et.al [2]. This scheme is widely adopted as it scales well for the larger groups. They employ a central, trusted *KDC* which manages the whole group. To manage the groups, it organizes the keys held by the users in the form of a tree called *key tree*. The group members are at the leaves of the *key tree*. Each group member is given a individual key (private key) that he shares with the *KDC*. *KDC* and the member can communicate using shared key. The *group key is at the root of the key tree*. The *KDC* distributes the *group key* to all the members securely. Along with the shared private key and the *group key* the group member is given the intermediate node keys along the path from leaf to root of the *key tree*. These intermediate node keys except shared private key and *group key* are called *auxiliary keys*. The *auxiliary keys* assist in *rekeying* process. Each user stores at most $O(log_2 n)$ keys.

There are schemes based on the Wong et al scheme aimed at reducing the *rekeying* cost. All the *group key management schemes* proposed address the single secure group (set of users forming a single group).

All the existing schemes address the issue of secure group key management in a single group. In [1] the scenario of simultaneous multiple groups with inter-group membership (we term as *overlapping membership*) is addressed based on *shamir's secret sharing scheme*.

In this work, we give a new approach to the group key management for multiple groups functioning simultaneously and having overlapping memberships among multiple groups. We propose a *Key-User Tree* structure for managing the group keys of multiple groups. Our scheme discussed this paper improves significantly over [1] in terms of *rekeying cost* and *storage*.

Rest of the paper is organized as follows. Section 2 introduces the multiple groups and overlapping membership concept. In Section 3 Lagrange form of interpolation and few lemma's are discussed. Notations and definitions are in Section 4. In Section 5, the proposed *Key-User Tree* (KUT) is discussed. The detailed approach, join and leave protocols and illustrative examples are elaborated in Section 6. Analysis, results and comparison with existing scheme are in section 7. Section 8 concludes followed by references.

II. Multiple Groups and Overlapping Membership

Multiple groups are the collection of subgroups which are groups on their own. Each subgroup comprises of a set of distinct users. Each subgroup is a secure subgroup. Members of the subgroup communicate using the respective *group key* of the group. So, the secure multiple groups is a collection of secure subgroups. In Fig 1 there are three groups *Group A, Group B*, and *Group C*. For distinguishing the users of the various groups , members of the groups are colored green, blue and red for *Group A, Group B*, and *Group C* respectively. In Fig 1 there are 8 users colored green in *Group A*. For these users *Group A* is the **parent group**. Likewise, there are 9 users each in **parent group** *Group B* and **parent group** *Group C* colored blue and red respectively. The secure groups *Group A, Group B,* and *Group C* operate simultaneously. Therefore, these are termed as *simultaneous multiple groups.*

Overlapping membership is defined as the members of *Group i* for whom the *Group i* is the *parent group* and want to communicate with members of other groups *Group j*, where *i* $\neq$ *j* and $i, j = 1, 2, 3$ in Fig 1.

In Fig 1 , the overlapping memberships can be interpreted as described below.

1) The overlapping memberships of the group members of *parent group A*,
    - One of the member of *Group A* (colored green) wants to communicate with (Group C). This can be seen in the area $A \cap C$ of Fig 1. So, that user is said to have a overlapping membership with the *Group C*.
    - Likewise, a member of *Group A* in $A \cap B$ has overlapping membership with the *Group B*.
2) The overlapping membership of the group members of *parent Group B*,
    - A member of *Group B* in $B \cap A$ has a overlapping membership with *Group A*.
    - A member of *Group B* in $B \cap C$ has a overlapping membership with *Group C*.

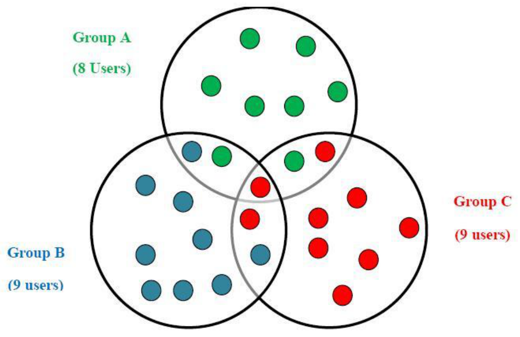

Fig. 1. Multiple Groups and Overlapping membership

3) The overlapping membership of the group members of *parent Group C*,
   - A member of *Group C* in $C \cap A$ has a overlapping membership with *Group A*.
   - A member of *Group C* in $C \cap B$ has a overlapping membership with *Group B*.
   - A member of *Group C* in $C \cap B \cap A$ has a overlapping memberships with both the groups *Group A* and *Group B*.

The group members who have the overlapping membership with *non-parental groups* (groups other than the group member's *parent group*) should be given the *group keys* of the *non-parental groups*. The group member with overlapping membership could communicate using these *group keys* with *non-parental group*.

Group key management in the concurrent multiple groups with overlapping membership is an issue to be addressed. In this work, we propose a group key management scheme in the multiple groups with overlapping membership.

III. The Lagrange form of the interpolation polynomial

Given a set of k points, $\{(x_1, y_1), \ldots, (x_j, y_j), \ldots, (x_k, y_k)\}$ where no two x_j are same, the *interpolation polynomial in the Lagrange form* is the linear combination,

$$P(x) = \sum_{j=1}^{k-1} y_j l_j(x) \tag{1}$$

of Lagrange's basis polynomials

$$l_j(x) = \prod_{1 \le i \le k, i \ne j} \frac{x - x_i}{x_j - x_i}$$

The degree of the polynomial $P(x)$ is less than or equal to $k-1$.

We use the fact that, given distinct set of k points, distinct unique polynomials are constructed using equation (1). To illustrate this fact, consider two sets of points

$$P_1 = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{k-1}, y_{k-1}), (x_k, y_k)\}$$

and

$$P_2 = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{k-1}, y_{k-1}), (x_m, y_m)\}$$

with the following properties. $|P_1| = |P_2| = k$. No two $x_i's$ in P_1 are same. The same holds for P_2. Let,

$$P_1 \cap P_2 = \{(x_1, y_1), \ldots, (x_{k-1}, y_{k-1})\} \ and \ |P_1 \cap P_2| = k-1$$

$$P_1 \cup P_2 = (P_1 \cap P_2) \cup \{(x_k, y_k), (x_m, y_m)\} \ and \ |P_1 \cup P_2| = k+1$$

$P_1 \cap P_2$ contains all the points common to both P_1 and P_2. By adding the point (x_k, y_k) to the set $P_1 \cap P_2$ and using (1) a polynomial $P_1(x)$ of degree $k-1$ can be constructed. Likewise, by adding the point (x_m, y_m) to $P_1 \cap P_2$ and using (1) another distinct polynomial $P_2(x)$ of degree $k-1$ can be constructed. So, the following lemma follows.

Lemma 1: Let $S = \{(x_1, y_1), \ldots, (x_{k-1}, y_{k-1})\}$ be a set of $k-1$ points where each x_i and $y_i, i = 1, \ldots, k-1$ is chosen from Galois Field $GF(p)$, where p is sufficiently large prime. In S no two $x_i's, i = 1, \ldots, k-1$ are same. Add a point (x_k, y_k), such that $x_k \ne x_j$, for all $j = 1, \ldots, k-1$ in S. Using Lagrange's form of interpolation given in (1) a polynomial $P(x)$ of degree at most $k-1$ can be constructed. For each distinct (x_i, y_i), $i = 1, \ldots, n$, not in S, n polynomials with degree at most $k-1$ can be constructed using (1). Therefore, n polynomials can be constructed using $n+k-1$ points.

Proof: The proof of the *Lemma 1* can be easily obtained using the Lagrange form of the interpolation polynomial given in (1). ■

IV. Notations and Definitions

- $U = \{u_1, \ldots, u_n\}$ is the set of n users. Let u_i be the *identity* of user i.
- $S_1, S_2, \ldots, S_m$ are the m groups comprising of the distinct subsets of users from U. For simplicity, we assume,

$$S_1 = \{u_1, \ldots, u_{\lfloor \frac{n}{m} \rfloor}\},$$
$$S_2 = \{u_{\lfloor \frac{n}{m}+1 \rfloor}, \ldots, u_{2\lfloor \frac{n}{m} \rfloor}\}$$
$$\ldots\ldots$$
$$S_n = \{u_{(m-1)\lfloor \frac{n}{m}+1 \rfloor}, \ldots, u_n\}$$

.

- $x \rightarrow y : z$ denotes, if y is a single user, sending a message z from user x to y. If y is a set of users sending a message z from x to set of users y (using unicast or multicast).
- $\{M\}_K$: Encrypt the message M with the key K. If $M = m_1, m_2, \ldots, m_n$, then encrypt each $m_i, i, \ldots, n$ and send as one message or encrypt $m_1, \ldots, m_n$ and send. In the latter it is assumed that receiver knows to segregate the decrypted message.
- *userset(K)* : It is the set of users who have key K.
- $u_k \rightarrow KDC : (J, S_i)$, Join request to KDC from a user u_k to join group S_i. A single user u_k can be replaced by a set of users $\{(u_1, \ldots, u_t)\}$, when the context is clear. This is applicable to rest of the definitions that follow.
- $u_k \rightarrow KDC : (L, S_i)$, Leave request to KDC from a user u_k whose parent group is S_i

- $u_k \to KDC : (J, S_i, S_j)$, It is a non-parental group join request. A user u_k whose parent group is S_i sends this request to join S_j.
- $u_k \to KDC : (L, \epsilon, S_j)$, It is a request to KDC from a user u_k who has already left the parent group and wants to leave the non-parental group S_j. ϵ specifies that u_k has left the parent group prior to this request.
- *Joining Point:* The node in the KUT at which the newly joined user is attached.
- *Parent group of a user :* Suppose a user u sends (J, S_i) request to KDC. Then, the group S_i is termed as the *parent group* of the u. Also, it is a group in which the joining point of a user is defined in the *right subtree* of the corresponding KUT of the group(Essentially,the joining point is in the LKT which is a *right subtree* of the KUT).
- *Non-parental groups of a user*: Suppose a user u sends (J, S_i, S_j) request to KDC. Then, the group S_i is the *parent group* of u and S_j is the $non-parental$ group of u. The groups in which the joining points of the user are defined at the left subtree of the respective KUT's of the groups.
- *Storage cost at a user*: The storage at a user is defined in terms of the *number of points or shares* used to construct the *group keys* and the number of *auxiliary keys*.

V. Proposed Key-User Tree Structure

We define a tree structure that is constructed by the KDC for managing the *group keys* in multiple secure group communication with overlapping membership scenario. We term this tree structure as *Key-User Tree* abbreviated as KUT. The KUT is constructed by the KDC for each secure group. The KUT is partially based on the *Logical Key Tree (LKT)* key tree proposed by wong et al [2].

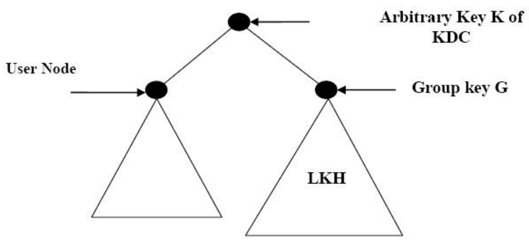

Fig. 2. Key-User Tree of the group S

In a multiple group with overlapping membership scenario, we categorize the users of a group into two.

1) *Parent group users*: Who initially join the group and the users who have only one joining point at the right subtree of the KUT.
2) *Non-parental group users*: Users joining from other groups.

The LKT is constructed as in [2] by the KDC for the *parent group users*. The *group key* G of the *parent group users* is the root of the constructed LKT. Suppose, t is the number of *parent group users*, then the height of constructed LKT is $\lceil log_2 t \rceil$.

Suppose, k is the number of *non-parental group users*. Let these users be $u_1^{'}, \dots, u_k^{'}$. KDC forms a *binary tree* with $u_i^{'}, i = 1, \dots k$ as nodes, $u_1^{'}$ being the root of *binary tree.*

KDC constructs the KUT as follows. Reader is instructed to refer Fig 2.

- It chooses an *arbitrary key* K. This is made as the *root* of the KUT.
- The *right child* of the *root* node K of KUT is the tree LKT constructed by the KDC rooted at the *group key* G of the group.
- The *left child* of the *root* node K of KUT is the binary tree of the user nodes of the non-parental groups. This left child subtree is rooted at any user node. In particular, it is the user node of the *non- parental user* who has sent the first *non-parental group* join request.

In Fig 2, the construction of KUT for an arbitrary group S is illustrated. Fig 3 illustrates the overlapping memberships between three groups.

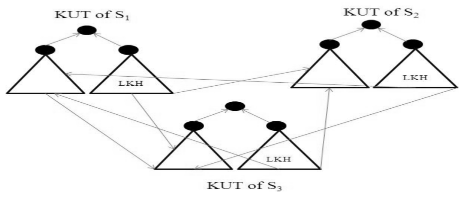

Fig. 3. Key-User Trees of 3 groups with overlapping memberships

A. Salient Features of the Key-User Tree

We note some of the properties of the *Key-User Tree* constructed by the KDC for an arbitrary group S.

1) ***Height of the*** KUT, h:
   - Suppose,there are t *parent group users* and k *non-parental group users*. If $k \leq 2t - 1$ then, $h = \lceil log_2 t \rceil + 1$.
   - If, $k > 2t - 1$, then $h = \lceil log_2 k \rceil + 1$
2) ***Effective operational height of KUT***: We define the *effective operational height* of KUT with height $\lceil log_2 t \rceil + 1$ as $\lceil log_2 t \rceil$. It is to be carefully noted that, the arbitrary key K is kept only by KDC, will never be distributed or used in managing the *group keys* of the *parent group of users* and *non-parental group users*. In our estimation of the cost, the additive constant one in h has no effect.
3) ***Joining point distinction in KUT by KDC:*** KDC defines the joining points of the join requests as below.
   - If $u \to KDC : (J, S)$, KDC finds the joining location only in the *right subtree of* KUT of S.
   - If $u \to KDC : (J, S^{'}, S)$, KDC finds the joining location only in the *left subtree of* KUT of S.
4) ***Relation between number of joining points and h:*** The KUT with $h = \lceil log_2 t \rceil + 1$ has atmost t joining points for the *parent group users* and atmost $2t - 1$ joining

points for *non parental group users.* Increasing h by 1 will define another t more joining points(for parental join requests) in the left subtree of KUT whereas, $2t$ more joining points are defined i.e a KUT of h + 1 will have $2t$ and $4t-1$ *parental* and *non-parental* joining points respectively.

VI. Our approach to multiple group key management scheme with overlapping membership

We give a scheme for managing the secure multiple groups with overlapping memberships. Our scheme is a centralized multiple group key management scheme. It employes a central, trusted entity called *Key Distribution Center (KDC)*. KDC manages the multiple secure groups allowing the overlapping membership. KDC uses the proposed *Key-User Tree (KUT)* to manage the group keys of multiple secure groups. The trusted entity KDC handles the join requests for parent group and non-parental group, leave request from a group member from its parent group and leave request from a non-parental group by a user. So, KDC carries out the *rekeying* process upon join/leave requests by the members. KDC manages the multiple secure groups by constructing the KUT for each secure group. In this section, we detail about the aspects of multiple secure group key management.

A. Group Setup and Group Key Distribution

- The security parameter k is chosen by the KDC. It chooses and fix $GF(p)$ for a chosen large prime.
- Initially, there are no users in any groups.
- Suppose, there is a set U of n users. They want to join the m groups,$S_1, \dots, S_m$ by sending the following requests.
  $\{u_1, \dots, u_{\lfloor \frac{n}{m} \rfloor}\} \rightarrow KDC : (J, S_1)$
  $\{u_{\lfloor \frac{n}{m}+1 \rfloor}, \dots, u_{2\lfloor \frac{n}{m} \rfloor}\} \rightarrow KDC : (J, S_2)$
  $\dots\dots\dots\dots$
  $\{u_{(m-1)\lfloor \frac{n}{m}+1 \rfloor}, \dots, u_n\} \rightarrow KDC : (J, S_m)$.
- KDC after receiving the join request, authenticates the user requests (we are not addressing the authentication process as it is not goal of the paper, but the KDC can use any of the existing ways of authenticating the users). For each user u_i, $i = 1, \dots, n$ generates a corresponding key K_i. K_i is the *private key* of the user u_i that KDC shares with u_i. K_i is given securely to u_i by KDC (Initially,we assume the existence of a secure channel between KDC and u_i). Using K_i, user u_i and KDC can securely communicate.
- KDC chooses $k-2$ points $(x_i, y_i), i = 1, \dots, k-2$. The x_i and y_i values for $i = 1, \dots, k-2$ are chosen randomly and independently from $GF(p)$ such that, no two $x_i's$ are equal. It means that all are distinct points. It chooses another point (x_{k-1}, y_{k-1}) such that $x_{k-1} \neq x_i$ for all $i = 1, \dots k-2$. We term the point (x_{k-1}, y_{k-1}) as *polynomial construction trigger share* (It should be noted that we use point and share interchangibly). KDC sends the points $(x_i, y_i), i = 1, \dots, k-2$ to all the users of groups $S_1, \dots S_m$. The points $(x_i, y_i), i = 1, \dots, k-2$ are termed as *prepositioned base shares*.
- After selecting $k-1$ points as described above, KDC selects m points $(x_{s_j}, y_{s_j}), j = 1, \dots, m$ by picking x_{s_j} and y_{s_j}, $j = 1, \dots, m$ from $GF(p)$ with the following constraints. All the m $x'_{s_j}s$ are distinct and no $x_{s_j}, j = 1, \dots, m$ is equal to any of the $x_i, i = 1, \dots k-1$ that are previously chosen from the KDC. The point (share) (x_{s_j}, y_{s_j}) is termed as *group specific share* of a user who is joining the group S_j.
- KDC constructs the tree KUT for each group $S_j, j = 1, \dots, m$ as described below:
  - KDC constructs LKT for the group S_j as in [2] with the difference that the *group keys* and the auxiliary keys are computed in different way. Auxiliary keys are computed as done in LKT[2], but the group keys are computed out of shares that are sent by KDC to all the users. This fact is explained in detail below. The leaves of the LKT are the user nodes which have sent join request for the group S_j.
  - The *group key* G_j of group S_j is computed by the KDC is as below. Using the shares $\{(x_1, y_1), \dots, (x_{k-2}, y_{k-2}), (x_{k-1}, y_{k-1}), (x_{s_j}, y_{s_j})\}$ and applying the *Lagrange form of interpolation polynomial* as given in (1) KDC constructs a polynomial $S_j(x)$. The polynomial $S_j(x)$ is evaluated at $x = 0$ and the value obtained is the *group key* G_j of S_j. Therefore, $G_j = S_j(x = 0)$.
  - KDC sends *auxiliary keys* of the constructed LKT to all the respective users.
  - Suppose the group S_j has t users, then, the height of the LKT is $\lceil log_2 t \rceil$. The number of auxiliary keys of a user in S_j is $\lceil log_2 t \rceil - 1$. The reader should notice carefully that, we are excluding the *group key* and the *shared private key* of the user in the LKT. Therefore, the *auxiliary keys* are represented as the intermediate nodes of the LKT. The LKT is rooted at G_j.
  - Now, KDC will construct the KUT rooted at K, where K is an arbitrary key which will not be given to any of the users of group G_j as described here. KDC makes LKT rooted at G_j constructed in the manner described above as the right subtree of the KUT rooted at K.
  - Initially, the *left subtree* of root node K is empty. The *left subtree* is populated with user nodes as the join requests come from the users who belong to other groups S_k, where $k \neq j$.
- KDC sends (x_{s_j}, y_{s_j}) to all the users who have sent request to join the group S_j where, $j = 1, \dots, m$.
- Now,a user who has sent a request to join the group S_j will have $k-2$ points (shares) which are *prepositioned base shares* $\{x_1, x_2, \dots, x_{k-2}\}$ and a (x_{s_j}, y_{s_j}) which is a *group specific share* of group S_j .
- KDC sends the *polynomial construction trigger share* (x_{k-1}, y_{k-1}) to all the users of group $S_j, i = 1, \dots, m$. Now each user of the group $S_j, i = 1, \dots, m$ has the

following shares.

- $(k-2)$ *prepositioned base shares*, $\{(x_1, y_1), \ldots, (x_{k-2}, y_{k-2})\}$.
- One group specific share (x_{s_j}, x_{s_j}) and
- One *polynomial construction trigger share* (x_{k-1}, y_{k-1})
.

- Using *Lemma 1* each user of group $S_j,\ j = 1, \ldots, m$ constructs a polynomial $S_j(x)$ of degree $k-1$. The polynomial $S_j(x)$ is evaluated at $x = 0$ to get the *group key* G_j of the group S_j.

Lemma 2: Let the number of users whose parent group is S_j be t. Each of the t user in S_j is given $(k-2)$ prepositioned base shares, one polynomial construction trigger share and one group specific share. Group key G_j is computed with these k shares using (1). Each of the t user $u_i, i = 1, \ldots, t$ is given a private key shared with the KDC and $\lceil log_2 t\rceil - 1$ auxiliary keys where $\lceil log_2 t\rceil$ is the height of the right subtree (LKT rooted at G_j) of the KUT of group S_j.

B. Join Events

In this section, we elaborate in detail the process of a user joining the multiple groups. The protocol or scheme for a *new parent group user* to join is given in *Protocol 1*. The scheme for a *non-parental group member* to join a group is given in *Protocol 2* .

C. Leave Events

In this section, we elaborate in detail the process of a user leaving the multiple groups.The protocol or scheme for a *new parent group user* to leave is given in *Protocol 3*. The scheme for a *non-parental group member* to leave group is given in *Protocol 4* .

D. Illustrating the multiple groups with overlapping membership with examples

To illustrate the *simultaneous groups with overlapping memberships*, join and leave operations we consider the following example scenario. Let, $S_1 = \{u_1, \ldots, u_7\} \bigcup \{u_9, \ldots, u_{13}\}$ and $S_2 = \{u_9, \ldots, u_{15}\} \bigcup \{u_1, \ldots, u_4\}$. $\{u_1, \ldots, u_7\}$ and $\{u_9, \ldots, u_{15}\}$ are the *parent group members* of S_1 and S_2 respectively. Users $\{u_9, \ldots, u_{13}\}$ of S_2 have *overlapping membership* with S_1. Likewise, $\{u_1, \ldots, u_4\}$ of S_1 have *overlapping membership* with S_2.

KDC constructs the KUT for S_1 as in Fig 4. It constructs , LKT for $\{u_1, \ldots, u_7\}$. KDC chooses arbitrary root key K_{s_1} for KUT. It makes LKT as *right subtree* of K_{s_1} and a *binary subtree* of users $\{u_9, \ldots, u_{13}\}$ is made as the *left subtree*. KDC constructs in the same way the KUT of S_2 by chosing the arbitrary key K_{s_2}. KUT of S_2 is as in Fig 5 .

1) Illustrating the join event: Consider the event u_8 joining S_1 (note that this is a parent group join as u_8 is not a *parent group member* of either S_1 or S_2). KDC finds the joining point as K_{7-8} , changes $K_{7-8}, K_{5-8}, and\ K_{1-8}$ (by choosing *new group specific share*) and sends the following. Refer Fig 4 and Fig 6.

Protocol 1: Join protocol for the parent group

```
1 $u \rightarrow KDC : (J, S_j)$ ;
2 $KDC \Longleftrightarrow u$ : Authenticate the user $u$ and distribute $K_u$
  // $K_u$ is the shared private key of the user $u$ with $KDC$
3 $KDC$: Finds the joining point in the right subtree ($LKT$) of $KUT$ of group $S_j$ and attaches $K_u$.
4 KDC : Changes the keys in the right subtree (LKT) of $KUT$ from joining point till the key node of the right child of root of $KUT$.
      // Essentially, till the root of the LKT. Note that the root of the $KUT$ is not changed;
5 Let $p_j$ be the joining point.
  $p_0$ the root of the right child of the root of the $KUT$
  for $i \leftarrow 1$ to $j$ do
      $p_{i-1}$ be the parent of $pi$
  end
  Let $K_{j+1}$ be $K_u$ and $K_0, \ldots, K_j$ be the old keys of $p_0, \ldots, p_j$ respectively.
  Let $(x_{s_j}^{'}, y_{s_j}^{'})$ be the old group specific share of group $S_j$.
  $KDC$ randomly generates new auxiliary keys $K_1^{'}, \ldots, K_j^{'}$.
  $KDC$ randomly chooses a new distinct group specific share $(x_{s_j}^{'}, y_{s_j}^{'})$ with the constraint that, $x_{s_j}^{'}$ is distinct from the $x_i^{'}s$ of prepositioned base shares, polynomial construction trigger share and the old group specific share of group $S_j$ ;
  for $i \leftarrow 1$ to $j$ do
      $KDC \rightarrow (userset(K_i) - userset(K_{i+1}))$: $\{(x_{s_j}^{'}, y_{s_j}^{'}), K_i^{'}\}_{K_i}$
  end
6 $KDC \rightarrow u$ : $\{\{(x_1, y_1), \ldots, (x_{k-2}, y_{k-2})\}, (x_{k-1}, y_{k-1}), (x_{s_j}^{'}, y_{s_j}^{'}), K_1^{'}, \ldots, K_j^{'}\}_{K_u}$;
7 Let , $\{u_1^{'}, \ldots, u_k^{'}\}$ be the user nodes at the left subtree of the $KUT$. $\{userset(K_0) - userset(K_1)\}$ contains all the users in the left subtree of $KUT$ and the users in the right subtree of the node $p_0$ whose key is $K_0$.
  $KDC \rightarrow \{userset(K_0) - userset(K_1)\}$ : $\{(x_{s_j}^{'}, y_{s_j}^{'})\}_{K_0}$;
8 New group key $K_0^{'}$ is computed by all the users of $S_j$ and the newly joined user $u$ using the Lagrange form of interpolation polynomial and Lemma 1 and evaluating the resultant polynomial at 0.
```

Protocol 2: Join protocol for the non-parent group

```
1 $u \rightarrow KDC : (J, S_i, S_j)$ ;
2 $KDC$ : Finds the joining point in the *left subtree* of the
  root node of $KUT$. $KDC$ should change only the *group
  key* of the $KUT$ of $S_j$ ;
3 Let $K_0$ be the group key of the group $S_j$;
4 $KDC \rightarrow \{userset(K_0)\} : \{(x_{s_j}^{'}, y_{s_j}^{'})\}_{K_0}$;
5 $KDC \rightarrow u : \{(x_{s_j}^{'}, y_{s_j}^{'})\}_{K_u}$;
  // $userset(K_0)$ includes all the users who
  have key $K_0$. It means all the parent
  group and non-parent group users of $S_j$;
6 The user $u$ and all the users in $userset(K_0)$ will get the
  new *group key* $K_0^{'}$. This is done using the *Lemma 1* and
  *lagrange for of polynomial construction*. The obtained
  polynomial is evaluated at 0 to get the *group key*.
```

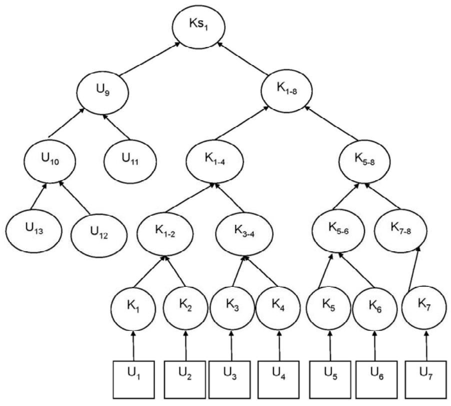

Fig. 4. KUT of Group S_1

$KDC \rightarrow \{u_1, \ldots, u_7\} \cup \{u_9, \ldots, u_{12}\} : \{(x_{s_1}^{'}, y_{s_1}^{'})\}_{K_{1-8}}$
$KDC \rightarrow \{u_5, u_6, u_7\} : \{K_{5-8}^{'}\}_{K_{5-8}}$
$KDC \rightarrow \{u_7\} : \{K_{7-8}^{'}\}_{K_{7-8}}$
$KDC \rightarrow \{u_8\} : \{(x_1, y_1), \ldots, (x_{k-1}, y_{k-1}), K_{5-8}^{'}, K_{7-8}^{'}\}_{K_8}$
All the users $u_1, \ldots, u_8$ and $u_9, \ldots u_{12}$ construct the new *group key* using $(x_{s_1}^{'}, y_{s_1}^{'})$. Changed keys are communicated to appropriate users. This is depicted in Fig 6 (from Fig 4. to Fig 6.)

Consider the event u_5 joining S_2. KDC finds the *joining point*at the *left subtree* of $S_2^{'}s$ KUT. KDC finds the new *group specific share* $(x_{s_2}^{'}, y_{s_2}^{'})$ and performs as below. Refer Fig 5 and Fig 7.
$KDC \rightarrow \{u_9, \ldots, u_{15}\} \cup \{u_1, \ldots, u_4\} : \{(x_{s_2}^{'}, y_{s_2}^{'})\}_{K_{9-16}}$
$KDC \rightarrow \{((x_{s_2}^{'}, y_{s_2}^{'}))\}_{K_5}$ All the users $u_9, \ldots, u_{15}$ and $u_1, \ldots, u_4$ compute the new *group key*. This is depicted in Fig 7 (change from Fig 5 to Fig 7).

2) Illustrating the leave event: Consider the event of u_6 leaving S_1 (u_6 leaving the KUT in Fig 6) then KDC changes the keys $K_{5-6}, K_{5-8}^{'}$ and $K_{1-8}^{'}$, chooses distinct $(x_{s_1}^{''}, y_{s_1}^{''})$. KDC does the following. Refer Fig 6 and Fig 8.
$KDC \rightarrow u_5 : \{K_{5-6}^{'}, K_{5-8}^{''}, (x_{s_1}^{''}, y_{s_1}^{''})\}_{K_5}$

Protocol 3: Leave protocol for the parent group

```
1 $u \rightarrow KDC : \{(L, S_j)\}_{K_u}$. $K_u$ is the *private key of* $u$ ;
2 $KDC$ finds the leaving point which is the parent of $K_u$;
3 Removes $K_u$ from the $KUT$. In particular, from $LKT$
  (right subtree of $KUT$);
4 Let $p_{j+1}$ denote the $K - node$ being deleted for $K_u$, $p_j$
  be the leaving point. $p_0$ be the root of the right child of
  the $KUT's$ root node. Let $K_0$ be the *group key* of $S_j$ ;
  **for** $i \leftarrow 1$ **to** $j$ **do**
      $p_{i-1}$ be the parent of $p_i$
  **end**
  $KDC$ generates randomly the new keys $K_1^{'}, \ldots, K_j^{'}$ for
  $p_1, \ldots, p_j$. $KDC$ randomly chooses a distinct *group
  specific share* $(x_{s_j}^{'}, y_{s_j}^{'})$ such that , $x_{s_j}^{'}$ is distinct from
  the $x_i$'s of the points chosen previously by the $KDC$.;
5 Let $K_{0,l}$ be left child keynode of $p_0$ $(K_0)$.
  $KDC \rightarrow userset(K_{0,l}) : \{(x_{s_j}^{'}, y_{s_j}^{'})\}_{K_{0,l}}$
  **for** $i \leftarrow 1$ **to** $j$ **do**
      **foreach** *child node* $y \neq p_{i+1}$ *of* $p_i$ **do**
          Let $K$ denote key at $k - node$
          $KDC \rightarrow userset(K)$ :
          $\{K_i^{'}, (x_{s_j^{'}}, y_{s_j^{'}}), K_{i-1}^{'}, \ldots, K_1^{'}\}_K$
      **end**
  **end**
6 Let $U_l = \{u_1, \ldots, u_k\}$ be the set of $k$ user nodes at the
  left subtree of $KUT$ of $S_j$
  **if** *There is a common shared key* $CK_l$ *such that,*
  $userset(CK_l) = \{u_1, \ldots, u_k\}$ **then**
      $KDC \rightarrow \{u_1, \ldots, u_k\} : \{(x_{s_j}^{'}, y_{s_j}^{'})\}_{CK_l}$
  **else**
      Let $U_l^{'} = U_l$;
      $i \leftarrow 0$;
      **repeat**
          Let, $U_{l,i+1}$ be the subset of users $U_l^{'}$. Let,
          $CK_{l,i+1}$ be the *common shared key* such that
          $userset(CK_{l,i+1}) = U_{l,i+1}$
          **if** *such* $U_{l,i+1}$ *exists* **then**
              $KDC \rightarrow U_{l,i+1} : \{(x_{s_j}^{'}, y_{s_j}^{'})\}_{CK_{l,i+1}}$
              $U_l^{'} = U_l^{'} - U_{l,i+1}$
          **end**
      **until** $U_l^{'}$ *is empty or there exist no subset of users*
      $U_{l,i+1}$ *with common key*;
  **end**
7 **if** $U_l^{'}$ *is empty* **then**
      **stop**
  **else**
      **foreach** $u$ *in the set* $U_l^{'}$ **do**
          $KDC \rightarrow u : \{(x_{s_j}^{'}, y_{s_j}^{'})\}_{K_u}$   // Where, $K_u$
          is the *private shared key* of user
          $u$
      **end**
  **end**
  ;
8 All the *parent group users* and *non-parent group users*of
  $S_j$ get the new *group specific share* and computes the
  new *group key* using (1) and *Lemma 1* ;
```

Protocol 4: Leave protocol for the non-parental group member

```
1 $u \rightarrow KDC : \{(L, S_i, S_j)\}_{K_u}$. or $u \rightarrow KDC : \{(L, \epsilon, S_j)\}_{K_u}$ $K_u$ is the *private key of* $u$.;
2 $KDC$ removes the user node $u$ from the *left subtree* of the $KUT$ ;
3 $KDC$ randomly chooses a distinct *group specific share* $(x_{s_j}^{'}, y_{s_j}^{'})$ such that , $x_{s_j}^{'}$ is distinct from the $x_i$'s of the points chosen previously by the $KDC$.;
4 Let $K_0$ be the *group key* of $S_j$. $K_0$ is the *right child* of the root of the $KUT$. Let, $K_l$ and $K_r$ be the *left* and *right child* of the $K_0$.;
5 $KDC \rightarrow userset(K_l) : \{(x_{s_j}^{'}, y_{s_j}^{'})\}_{K_l}$
6 $KDC \rightarrow userset(K_r) : \{(x_{s_j}^{'}, y_{s_j}^{'})\}_{K_r}$
7 $KDC$ follows the *step 6* of *Protocol* 3 to distribute $(x_{s_j}^{'}, y_{s_j}^{'})$ to the *non-parental group users*.
8 All the *parent group users* and *non-parent group users* of $S_j$ get the new *group specific share* and computes the new *group key* using (1) and *Lemma 1* ;
```

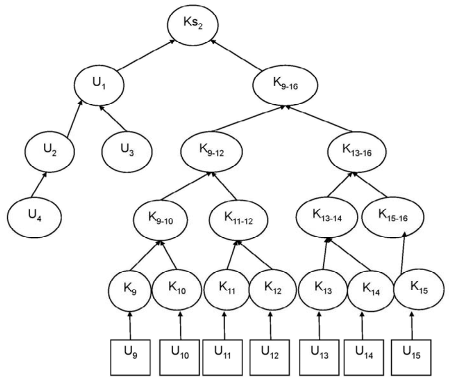

Fig. 5. KUT of Group S_2

$KDC \rightarrow \{u_7, u_8\} : \{(x_{s_1}^{''}, y_{s_1}^{''}), K_{5-8}^{''}\}_{K_{7-8}^{\prime}}$
$KDC \rightarrow \{u_1, \ldots, u_4\} : \{(x_{s_1}^{''}, y_{s_1}^{''})\}_{K_{1-4}}$
$KDC \rightarrow \{u_9, \ldots, u_{12}\} : \{(x_{s_1}^{''}, y_{s_1}^{''})\}_{K_{9-12}}$
$KDC \rightarrow u_3 : \{(x_{s_1}^{''}, y_{s_1}^{''})\}_{K_{13}}$
All the members of S_j construct the new *group key* and changed keys are sent to appropriate users. This event is depicted in Fig 8 (change from Fig 6 to Fig 8).

Consider the event of u_5 leaving S_2 (note *non-parent group member* leave). Now only the *new group specific share* should be chosen by KDC and sent. Refer Fig 7 and Fig 9.
$KDC \rightarrow \{u_9, \ldots, u_{12}\} : \{(x_{s_2}^{''}, y_{s_2}^{''})\}_{K_{9-12}}$
$KDC \rightarrow \{u_{13}, \ldots, u_{15}\} : \{(x_{s_2}^{''}, y_{s_2}^{''})\}_{K_{13-16}}$
$KDC \rightarrow \{u_1, \ldots, u_4\} : \{(x_{s_2}^{''}, y_{s_2}^{''})\}_{K_{1-4}}$
All the users will get the new *group key* no *auxiliary keys* are being changed. This is depicted in Fig 9 (change from Fig 7 to Fig 9).

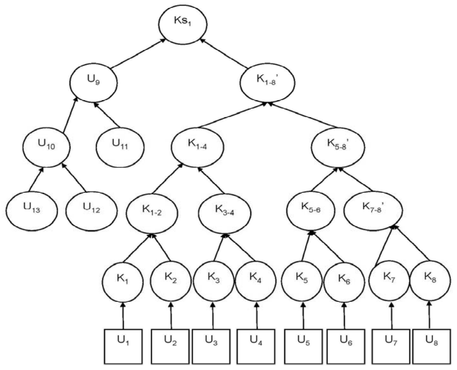

Fig. 6. KUT of Group S_1 after u_8 joins

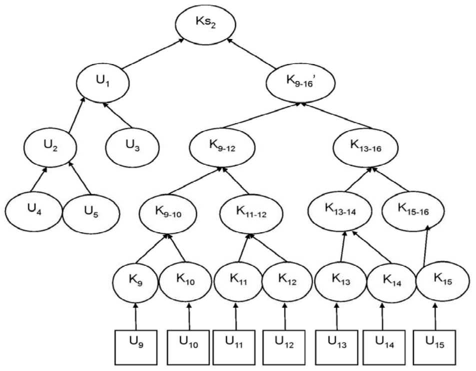

Fig. 7. KUT of Group S_2 after u_5 joins

VII. Analysis, Results and Comparison

A. Analysis of Group Join Protocols

1) **Number of encryptions**
   a) **Parent group join:** When the user joins the parent group with n *parent group users* and any number of *non-parent group users* , the number of encryptions performed are atmost $\lceil log_2 n \rceil + 1$.
   b) **Non-Parent group join**: When the *non-parent group user* joins the group with any number of *parent group users* and any number of *non-parent group users*, the number of encryptions performed is a constant 2.
2) **Number of key changes**:
   a) **Parent group join:** If there are n *parent group users* and any number of *non-parental group users*, the number of key changes upon a *parent group user* join is atmost $\lceil log_2 n \rceil$.
   b) **Non-Parent group join:** Irrespective number of *parent group users* and *non-parental group users* , the number of keys changed upon *non-parent group user* join is constant 1.
3) **Number of Rekey-Messages**:
   a) **Parent group join:** The number of re-key messages constructed upon a *parent group user* join

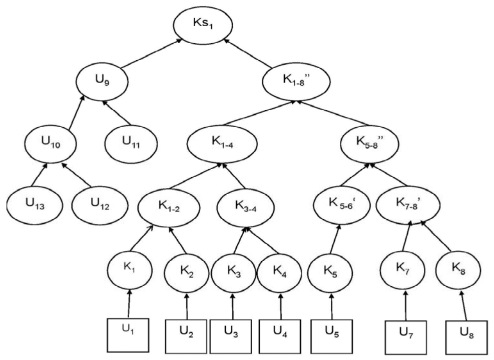

Fig. 8. KUT of Group S_1 when u_6 leaves

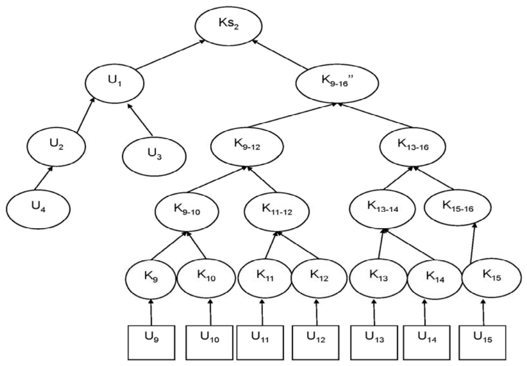

Fig. 9. KUT of Group S_2 when u_5 leaves

is atmost $\lceil log_2 n \rceil + 1$, where n is the number of *parent group users*.

b) **Non-Parent group join:** The number of re-key messages constructed upon *non-parental group user* join is a constant 2.

B. Analysis of Group Leave Protocols

1) **Number of encryptions**
   a) **Parent group leave:** When the *parent group user* leaves the *parent group* with n *parent group users* and t *non-parent group users* ,
   # of encryptions performed $\leq 2 \lceil log_2 n \rceil + t$. Reader should note that the # of encryptions ranges over $\{2\lceil log_2 n \rceil + 1, 2\lceil log_2 n \rceil + 2, \ldots, 2\lceil log_2 n \rceil + t\}$
   b) **Non-Parent group leave**: When the *non-parent group user* leaves the group with n *parent group users* and t *non-parent group users*,
   # of encryptions performed $\leq t + 2$.
   The *# of encryptions ranges over* $\{3, 4, \ldots, t+2\}$
2) **Number of key changes**:
   a) **Parent group leave:** If there are n *parent group users* and t *non-parental group users*, upon *parent group user* leave,
   # of keys changed $\leq \lceil log_2 n \rceil$ (This comprises of $log_2 n - 1$ *auxiliary keys* and one *parent group specific key*).
   b) **Non-Parent group leave:** Irrespective of number of *parent group users* and *non-parental group users* , upon *non-parent group user* leave,
   # of keys changed = 1.
3) **Number of Rekey-Messages**:
   a) **Parent group leave:** When the *parent group user* leaves the *parent group* with n *parent group users* and t *non-parent group users* ,
   # of re-key messages constructed $\leq \lceil log_2 n \rceil + t$.
   The # of re-key messages ranges over $\{\lceil log_2 n \rceil + 1, \lceil log_2 n \rceil + 2, \ldots, \lceil log_2 n \rceil + t\}$
   b) **Non-Parent group leave:** When the *non-parent group user* leaves the group with n *parent group users* and t *non-parent group users*,
   # of re-key messages constructed $\leq t + 2$.
   The # of re-key messages ranges over $\{3, 4, \ldots, t+2\}$

C. Storage Cost Estimation

We measure the *storage cost* in terms of the following parameters.

- Number of shares
- Number of auxiliary keys

We term combindly the both as *key material.*

We consider the following *three* categories of the users and estimate the storage cost.

1) **User of a *parent group* without any *overlapping membership***: The user u who is a member of *parent group* with n *parent group users*, t *non-parent group users* and not having any *overlapping memberships* with any other *non-parental groups* will store the following . *Key material=* ***(k-2)*** *prepositioned base shares +* ***1*** *polynomial construction trigger share +* ***1*** *group specific share of the parent group +* $\lceil log_2 n \rceil - 1$ *auxiliary keys + u's private shared key.* **Therefore, the *key material* stored at the user** u **=** $k + \lceil log_2 n \rceil$
2) **User of a *parent group* with** m ***overlapping memberships*:** The user u who is the member of the *parent group* with n *parent group users*, t *non-parent group users* and having m *overlapping memberships* with other m *non-parental groups* will store the following. **Key material=** ***(k-2)*** *prepositioned base shares +* ***1*** *polynomial construction trigger share +* ***1*** *group specific share of the parent group +* $\lceil log_2 n \rceil - 1$ *auxiliary keys + u's private shared key +* m *group specific shares of other* m *groups with which u has overlapping memberships.* **The key material stored at** $u = (k + m) + \lceil log_2 n \rceil$
3) **User who has left it's *parent group* and has** m ***overlapping memberships***: The user u who has left it's *parent group* will not have any *auxiliary keys*. As evident, u will store the following. **Key material=** ***(k-2)*** *prepositioned base shares +* ***1*** *polynomial construction trigger share + u's private shared key +* m *group specific shares of other* m *groups with which* u *has*

TABLE I
COMPARISON OF THE SCHEME IN [1] AND OUR PROPOSED KUT BASED SCHEME

	Scheme in [1]		Our Scheme based on KUT	
	# of encryptions	# of Key Changes	# of encryptions	# of Key Changes
Join of a parent group user	$2\lceil log_2 mn\rceil$	$\lceil log_2 mn\rceil$	$2\lceil log_2 n\rceil + 1$	$\lceil log_2 n\rceil$
Join of a non-parent group user	$2\lceil log_2 mn\rceil$	$\lceil log_2 mn\rceil$	2	1
Leave of a parent group user	$2\lceil log_2 mn\rceil$	$\lceil log_2 mn\rceil$	$2\lceil log_2 n\rceil + m - 2$	$\lceil log_2 n\rceil$
Leave of a non-parent group user	$2\lceil log_2 mn\rceil$	$\lceil log_2 mn\rceil$	$\leq (m + 2^{\frac{log_2 n-1}{2}})$	1
Storage at a user	$(m + k - 1)$ shares and $mlog_2 n$ auxiliary keys		$(m + k - 1)$ shares and $log_2 n$ auxiliary keys	

overlapping memberships. ***So, the key material stored at u =*** $(k + m) + 1$.

D. Significant results of the scheme based on KUT

Suppose there are n users and m groups $S_i, i = 1, \dots, m$. Each group $S_i, i = 1, \dots, m$ consists of $\lfloor \frac{n}{m} \rfloor$ *parent group users*.

- **Result 1:** Let, each of $\lfloor \frac{n}{m} \rfloor$ members of $(m - 1)$ groups are having overlapping membership with group $S_j, where\ j \neq i, i = 1, \dots, m$. So, upon a *parent group user* leave from S_j, the number of encryptions required to distribute the new *group specific share* $(x^{'}_{s_j}, y^{'}_{s_j})$ to all members having membership with S_j is $\lceil log_2 \lfloor \frac{n}{m} \rfloor \rceil + (m - 1)$.
- **Result 2:** Let t be the number of *non-parent group users* of some group S_j. Suppose, a *parent group member* of S_j leaves S_j, then the *number of encryptions* required to send the changed *group specific share* $(x^{'}_{s_j}, y^{'}_{s_j})$ to t *non-parent group users* follows,
$$\lim_{t \to (m-1)\lfloor \frac{n}{m} \rfloor} \#\ of\ encryptions = m - 1$$
- **Result 3:** Let t be the number of *non-parent group users* of some group S_j. Suppose, a *parent group member* of S_j leaves S_j, then the *number of encryptions* required to send the changed *group specific share* $(x^{'}_{s_j}, y^{'}_{s_j})$ to the *parent group users* of S_j is **2**. Interestingly, if l be the number of *parent group users* such that
$$l = 2^x, x = 1, 2, \dots \#\ of\ encryptions\ required = 2$$
- **Result 4:** Irrespective of number of *parent group users* and *non-parent group users* in a group, the number of encryptions performed when a *new parent group user* joins is atmost $\lfloor log_2 n \rfloor + 1$ and is 2 when a *new non-parent user* joins.
- **Result 5:** The storage at a user of a group who has membership with m groups is $(m + k - 1)$ shares and $log_2 l$ auxiliary keys. Where, l is the number of *parent group users* in the group.

E. Comparison with the existing approach

We compare our scheme with the scheme in [1]. Suppose, there are m groups and each group has n *parent group users* and each of the *parent group members* of every group has a overlapping membership with every other group. Therefore in a group, there are $(m - 1)n$ *non-parent group members* and n *parent group members*.The results of comparison based on this scenario are provided in *Table 1.* As it is depicted in the table our KUT based scheme out performs the scheme in [1].

VIII. CONCLUSION

A new approach to secure group keys management for simultaneous multiple groups with overlapping memberships is discussed. *Key-user Tree* is proposed for the group keys management for simultaneous multiple groups. A member of a group with n *parent group users* having overlapping membership with $(m-1)$ other groups stores only $(m+k-1)$ points/shares from $GF(p)$(where k is a security parameter) and atmost $\lceil log_2 n \rceil - 1$ *auxiliary keys*. A group with n *parent group users* having all members of *(m-1)* other groups as *non-parent group users* incurs only $2\lceil log_2 n \rceil + m - 1$ encryptions upon leave activity by a *parent group user* of the group. If a *non-parent group user* leaves atmost $(m + 2^{\frac{log_2 t-1}{2}})$ (t is the number of *parent group users* of *parent group* of leaving user) encryption operations are carried out.These results obtained by the proposed scheme depicts that the scheme scales well when there is increase in the overlapping memberships for a group. In comparison with the previous work of [1] our scheme is efficient.

ACKNOWLEDGMENT

This work was supported by *Ministry of Human Resource Development,* Government of India.

REFERENCES

[1] R. Aparna and B. B. Amberker, "Key management scheme for multiple simultaneous secure group communication," in *IEEE International Conference on Internet Multimedia Systems Architecture and Applications (IMSAA-09))*, Banglore, India, Dec. 2009.
[2] C.K.Wong, M. Gouda, and S. Lam, "Secure group communication using key graphs," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 16 – 30, Feb. 2000.
[3] C. Blundo, A. D. Santis, A. Herzberg, U. V. S. Kutten, and M. Yung, "Perfectly secure key distribution for dynamic conferences," in *Advances in Cryptology-CRYPTO'92,LNCS-740*, 1993, pp. 471–486.
[4] S.Mittra, "Iolus: A framework for scalable secure multicasting." in *Proceedings of the ACM SIGCOMM*, vol. 27, Newyork, USA, Sep. 1997, pp. 277–288.
[5] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Secure group communication using robust contributory key agreement," *IEEE Transactions on Parallel and Distributed System*, vol. 15, pp. 468–480, 2004.
[6] M. Burmester. and Y. Desmedt, "A secure and efficient conference key distribution system,," in *Advances in Cryptology - EUROCRYPT'94*, 1994.
[7] S. Rafaeli and D. Hutchison, "A survey of key management for secure group communication," *ACM Computing Surveys*, vol. 35, pp. 309 – 329, Sep. 2003.