MIS Review Vol. 18, No. 1, September (2012), pp. 71-91

Key Management Scheme for Cumulative Member Removal and Bursty Behavior in Secure Group Communication Using *m-ary* Tree

R. Aparna[1], B.B. Amberker[2]
[1]*Deptartment of Computer Science and Engineering, Siddaganga Institute of Technology*
[2]*Department of Computer Science and Engineering, National Institute of Technology*

ABSTRACT: *Secure group communication is an important research area and numerous applications are relied upon secure group communication model. Since the group is dynamic in nature, rekeying must be carried out in an efficient manner. Member leave event should be handled carefully compared to member join event. In some applications like pay-per-view, periodical electronic information distribution etc., many users join and leave the group at the same moment known as bursty behavior. In this paper, we propose schemes for handling cumulative member removal and bursty behavior. We use m-ary key tree for managing the secure group and maintain only m keys at each level of the key tree. We start with a scheme for cumulative member removal and then we handle all the possible bursty behavior scenarios. We analyze the communication and computation costs for worst cases. We compare the costs of our scheme with the schemes proposed by Li et al. (2001) and binary key tree scheme of Zou, Magliveras, and Ramamurthy (2002). We show that in our scheme the number of new keys generated and encryptions performed are less compared to Li et al. (2001) and Zou, Magliveras, and Ramamurthy (2002) schemes.*

KEYWORDS: *Secure Group Communication, m-ary Key Tree, Key Distribution Center, Cumulative Member Removal, Bursty Behavior, Encryption Keys.*

1. Introduction

With the widespread availability of powerful communication networks and explosive growth of Internet technologies, group communication has become a topic of considerable interest. Secure Group Communication (SGC) deals with exchange of information securely among a set of users. Many solutions have been proposed for key management in SGC. A naive approach is to have a common key shared between each pair of users in a group of size N. This results in storing $(N - 1)$ keys with each user and performs $(N - 1)$ encryptions to send a message. This scheme is proportional to N and does not scale well for a secure group with large number of users. Therefore, a better approach for SGC is to have a common secret key known as *group key* that is shared among members of the group.

Group membership may change over time. New members may join the group and existing members may leave the group. Whenever there is a membership change, it is necessary to change the group key to provide continued privacy. The process of changing the group key and communicating it to group members securely is known as *rekeying*. When a new member joins the group, the group key must be changed to avoid the new user from accessing past communication. This is known as *backward access control*. When a member leaves the group, the group key must be changed to avoid leaving member from accessing future communication. This is known as *forward access control*. A scalable SGC model ensures that the rekeying operation is carried out with minimal computation and communication costs.

Many applications like scientific discussion, teleconferencing, real-time information services, pay-per-view, etc., are based on SGC model. In applications like pay-per-view, periodical electronic information distribution, accessing online journals etc., many users subscribe and/or unsubscribe for the service at the same time. For instance, in the pay-per-view application, users subscribe at the start of a particular show and unsubscribe after the telecast of the show (for example sports event). We can have scenarios like single user join, single user leave, multiple users join, multiple users leave and sometimes a single user may join and a single user may leave the group at the same time. Similarly multiple users join and leave scenario may occur at approximately the same moment. This scenario in which multiple users join and/or multiple users leave is termed as *bursty behavior*. Whereas the scenario in which only multiple users leave, but no user joins is termed as *cumulative member leave* or *cumulative member removal*. The above mentioned applications exhibit bursty behavior and cumulative member leave. Hence, it is important to develop an efficient key management scheme to address these issues.

Many key management schemes have been proposed to manage SGC (Blundo et al., 1993; Burmester & Desmedt, 1995, 1997; Lee, Lui, & Yau, 2002; Lin, Lai, & Lee, 2005; Perrig, 1999; Rafaeli & Hutchison, 2003; Sherman & McGrew, 2003; Wong, Gouda, & Lam, 1998). In some schemes (Blundo et al., 1993; Fiat & Naor, 1993; Lin et al., 2005; Mittra, 1997; Perrig, 1999; Rafaeli & Hutchison, 2003; Sherman & McGrew, 2003; Wallner, Harder, & Agee, 1999; Wong et al., 1998) trusted *Key Distribution Center* (KDC) may be involved which generates and distributes initial private pieces of information to users in the group. Group key may be generated and distributed by the KDC itself or group members may compute the group key either interactively or non-interactively. The other category of schemes termed as *Contributory Key Agreement schemes* (Amir, Danilov, et al., 2004; Amir, Kim, et al., 2004; Burmester & Desmedt, 1995, 1997; Diffie & Hellman, 1976; Lee et al., 2002; Perrig, 1999) do not involve KDC, instead, all the group members contribute an equal share to compute the common secret key.

But, all the above mentioned schemes concentrate on reducing the number of encryptions and rekey messages when a single member joins or leaves the group. However, in Caronni et al. (1998), Chang et al. (1999), authors try to address the problem of multiple leave operations. In Li et al. (2001), joins and leaves are accumulated for a fixed period of time and then handled all at once, thereby reducing the frequency of key distribution. This is also considered as a case of bursty behavior. In Zou, Magliveras, and Ramamurthy (2002), Zou, Magliveras, and Ramamurthy address bursty behavior by extending the key management scheme proposed by Blundo et al. (1993). Here the storage required by each user is $\binom{n+t-1}{n-1}$ and the time complexity for computing the group key is $O(t^{2n})$, where n denotes number of users in the group and t, the threshold value. In Zou, Ramamurthy, and Magliveras (2002), bursty behavior is addressed by indexing users and keys with a binary representation and using binary right shift operation. The time complexity of this scheme is logarithmic in the group size.

The scheme proposed in Chang et al. (1999) focuses on the problem of cumulative member removal and finds the minimum number of messages required to distribute new keys to the remaining members of the group. The scheme uses a binary tree structure and assigns only two keys at each level of the tree. It finds the keys required for encrypting the changed keys by using Boolean function minimization technique. In Poornima, Aparna, and Amberker (2007a, 2007b) the scheme proposed in Chang et al. (1999) has been extended to *m-ary* tree and authors propose algorithms for single leave, two members leave and multiple members leave events. It handles only simultaneous leave operations.

In this paper, we extend the scheme proposed in Poornima et al. (2007a, 2007b) to address cumulative member removal and bursty behavior in SGC. To address cumulative member removal, we find encryption keys required to convey new group key to the remaining members of the group. We compare our scheme with Chang et al. (1999) scheme and show that we reduce the storage at KDC by a factor of 4% to 12%, at users by 30% to 38% and encryption cost is reduced by 55%. For bursty behavior scenario, we compute the number of keys generated, encryptions performed and rekey messages constructed for the worst case scenario. We address equal number of join and leave events, number of joins greater than number of leaves and number of joins less than number of leaves separately. We compare our results with the batch rekeying scheme proposed in Li et al. (2001) and show that the number of new keys generated and encryptions performed are either less than or equal to the values obtained in Li et al. (2001).

Rest of the paper is organized as follows: We discuss in brief the related work in Section 2, we begin with discussing the scheme for handling cumulative member removal in Section 3. In Section 4 we analyze bursty behavior with equal number of joins and leaves, number of joins more than number of leaves, number of joins less than number of

leaves scenarios. Section 5 highlights Implications for Practices and Section 6 concludes the paper.

2. Related work

The key management scheme for managing secure group communication proposed by Wong et al. (1998), Wong and Lam (2000) is one of the efficient method which employs a logical *key tree* structure. This scheme is also termed as Key Tree scheme. A key tree is a directed acyclic graph with two types of nodes, namely *U*-nodes representing group members and *K*-nodes representing keys. In this scheme KDC maintains a key tree with degree *d*, where each node in the tree corresponds to either member's private key, or group key or an auxiliary key. Users are at the leaf level of the tree and the key nodes at leaf level correspond to private keys of the users. The key at the root node of the tree is the *group key*. Keys at the internal nodes of the tree are the *auxiliary keys*. A member *u* of *U* is given a key *k* if and only if there is a directed path from *U*-node *u* to *K*-node *k* along the path from *U*-node to root. It addresses the problem of minimizing the number of rekey messages during single join or leave event. This scheme achieves scalable rekeying, which requires $2log_dN$ rekeying overhead for user join event and $(d - 1)\ log_dN$ for user leave event, where *N* is the group size. The scheme proposed in Waldvogel et al. (1999) reduces the overhead during user join event by allowing the users to calculate new key on their own using one-way function. The scheme proposed by Sherman and McGrew (2003) uses a one-way function tree in which keys are generated using one-way functions. In both the schemes rekeying overhead is reduced to log_2N instead of $2log_2N$.

In Li et al. (2001), Li et al. accumulate joins and leaves for a fixed period of time and then handle them in a batch termed as batch rekeying, thereby reducing the frequency of key distribution. A marking algorithm has been devised to process a batch of join and leave requests and the scheme is analyzed for worst case scenarios.

Zou, Magliveras, and Ramamurthy (2002) address bursty behavior using a symmetric polynomial in *N* indeterminates with coefficients in *Zp*, where *p* is a prime and $p \geq N$. It extends the scheme proposed by Blundo et al. (1993) and provides provision for accommodating users in the range *N* to 2*N*.

Bursty behavior is addressed in Zou, Ramamurthy, and Magliveras (2002) by using a key tree structure as in Wong et al. (1998). Here, a node at level *l* in the key tree is assigned with bitstring of length *l*, $1 \leq l \leq log_2N$. Each user u_i, $i = 1, 2, \ldots, N$ in the group is also assigned with an identification number *i*. To handle bursty behavior, it first finds the common keys shared among the leaving members by performing right shift operation on the bitstrings. It changes the common keys only once thereby reducing the computation

and communication costs. The amount of storage required at each user and KDC in this scheme is same as that of Wong et al. scheme (Wong et al., 1998).

The scheme in Chang et al. (1999) is proposed to handle multiple user leave event in large groups. The scheme uses a binary key tree to manage SGC. At every level i of the binary key tree a key pair (k_i, k'_i) is maintained, $i = 1, 2, ..., [log_2 N]$. Here, k_i and k'_i are not complement to each other, but are two different values. This scheme uses Boolean Function Minimization technique to compute minimum number of encryption keys to securely distribute the changed keys to the remaining users in the group. In Poornima et al. (2007a, 2007b) *m-ary* tree is used to manage SGC. Here, m keys are used at each level of the tree. It addresses single member leave, two members leave and multiple members leave events, thus handling only simultaneous leave operations.

The scheme in Zou, Ramamurthy, and Magliveras (2002) addresses only bursty behavior, but not cumulative member removal, whereas the scheme in Chang et al. (1999) addresses only cumulative member removal, but not bursty behavior. The scheme proposed in Poornima et al. (2007a, 2007b) also addresses only cumulative member leave event. In this paper, we are extending the scheme proposed in Poornima et al. (2007a, 2007b) to address both cumulative member removal and bursty behavior efficiently. To the best of our knowledge we found no paper in the literature which addresses both cumulative member removal and bursty behavior. In this paper we are addressing both cumulative member removal and bursty behavior by considering all the possible worst case scenarios in an efficient way.

3. Cumulative member removal

Chang et al. scheme proposed in Chang et al. (1999) uses a binary key tree and two keys are maintained at each level of the tree. Here, we extend the scheme to *m-ary* key tree as in Poornima et al. (2007a, 2007b) and maintain m keys at each level of the key tree. For a key tree with degree m, the height h of the tree is $log_m N$. We use the following model in the paper.

3.1 Model

We consider a group with N users $u_1, u_2, ..., u_N$. The tree is constructed with the following features:

(1) There are $h = \lceil log_m N \rceil$ levels in the tree and are numbered from 0 to h. The root is at level 0.

(2) Each node at level l can have atmost m children nodes, $l = 0, 1, ..., h - 1$.

(3) The children of a node are numbered from 0 to $m - 1$ from left child to right child. We refer these numbers as positions of the children nodes.

(4) Leaf nodes that are rooted at level l form one subgroup, $l = 0, 1, ..., h$. Subgroups at level l are numbered from 0 to $m^l - 1$.

(5) The key at level 0 is the group key GK.

(6) At each level l of the tree, m keys are assigned and are labeled as K_l^0 to K_l^{m-1}, $l = 1, 2, ..., h$.

(7) All the m keys at level l are assigned with unique Key Identification number, KID.

(8) All the nodes at position i of level l are assigned with the common key K_l^i, $l = 1, 2, ..., h$.

(9) Nodes at level 1 which are numbered as 0 to $m - 1$ are assigned the corresponding binary value as UIDs. The length of each UID is $[log_2 m]$ bits.

(10) Each node k at level l is assigned with UID BX, where X is the binary value of k and B is the UID of its parent, $l = 2, 3, ..., h$.

KDC generates and sends a private key K_i, $i = 1, 2, ..., N$ to each user u_i. We assume that KDC establishes a secure channel with each user u_i to communicate the private key K_i, $i = 1, 2, ..., N$. To each user u_i, KDC sends the keys and their KIDs along the path from leaf to root by encrypting with the private key of u_i.

3.2 Notations

(1) GK denotes Group Key.

(2) $\{GK\}_{K_0}$ denotes GK is encrypted with the key K_0.

(3) $||$ denotes concatenation operation.

(4) $\cup$ denotes set union operation.

3.3 Advantages of using m-ary tree

For a group with N users, if binary tree is used, it results in a tree of height $log_2 N$. Each user in the group is required to store $log_2 N$ keys and the KDC is required to store $2N - 1$ keys in the scheme proposed by Wong et al. (1998). In Chang et al. (1999), the storage at the KDC is reduced by using only 2 keys at each level, thus reducing the storage to $2log_2 N$. However the storage at each user remains as $log_2 N$. In our scheme, instead of binary tree, *m-ary* tree ($m > 2$) is used which reduces the height of the tree to $log_m N$ and hence, each user is required to store $log_m N + 1$ keys and KDC is required to store $mlog_m N + 1$ keys.

3.4 Optimal value for m

For a group with N users, if h is the height of the binary tree, it results in storing $2h$ keys at the KDC in the scheme proposed by Chang et al. in Chang et al. (1999). For the same value of N, if h' is the height of the *m-ary* tree, then mh' keys are stored at the KDC. The height h of the *m-ary* tree is

$$h' = h/log_2 m$$

Number of keys at the KDC in terms of h is expressed as $m(h/log_2 m)$, which illustrates that as m increases, the storage at the KDC increases. Hence, in order to reduce the number of keys both at the KDC and users as compared to the scheme in Chang et al. (1999), following relation must be satisfied:

$$m(h/log_2 m) < 2h$$

which is true only if $m = 3$ and $m = 4$. If $m = 4$ it maintains the same storage at KDC as in Chang et al. (1999) scheme and reduces the storage at the users. However if $m = 3$, it reduces the storage both at the KDC and users as compared to Chang et al. (1999) scheme. We achieve storage savings of 4% to 12% at KDC and about 30% to 38% at the users if we use $m = 3$.

3.5 Scheme for handling cumulative member removal

Any number of users can leave the secure group from any position in the *m-ary* tree. Algorithm 1 handles the computation of encryption keys for cumulative removal of arbitrary members. We use the following notations in the algorithm.

(1) L: Number of leaving users

(2) P: Array with L elements containing UIDs of leaving users

(3) *KEK*: Set containing the keys used to encrypt the new group key

(4) S: Set of users in the group excluding leaving members

Algorithm 1 considers L leaving users and finds a set of encryption keys to convey new group key to the remaining members in the group. If there is no leave in a subgroup rooted at level l, it is termed as *non-leaving subgroup*, $l = 0, 1, ..., h - 1$. In Step 1 of the algorithm, KDC finds non-leaving subgroups in the tree and appends the keys of the non-leaving subgroups to the set *KEK*. KDC starts from level 1, if there is a non-leaving subgroup j at level 1, the key K_1^j is added to the *KEK* and the users in subgroup j are eliminated from set S. This may result in eliminating the users belonging to the subgroups at subsequent levels of the tree. This procedure is repeated for all the levels from 2 to $h - 1$ in the tree for the remaining subgroups. When we move to Step 2, we are left with the

users who do not belong to any non-leaving subgroup. We have to find encryption key individually for each remaining user in set S. We compare the keys held by each user with the keys of all leaving members and consider the combination of keys as follows:

(1) In i^{th} iteration of the loop, we compare i keys of u_i from level h to $h - i$, $i = 2, 3, ..., h - 1$ with the respective i keys of leaving members. If no leaving member shares the same set of keys, we do not proceed with the next iteration.

(2) We compute the Ex-OR of the set of keys obtained from previous step.

To convey the new group key GK' securely to the remaining members of the group, KDC performs the following operations:

Algorithm 1: Computation of Encryption Keys by KDC for Cumulative Member Leave.

Input: Set S of users in the group excluding leaving members. Array P containing UIDs of leaving users.

Output: Set KEK of Encryption Keys.

1: **Step 1:** KEK $\leftarrow \phi$

2: **for** $l \leftarrow 1$ to h **do**

3: **for** $j \leftarrow 1$ to L **do**

4: $a \leftarrow l[log_2 m]$ MS bits P[j]

5: $b \leftarrow [log_2 m]$ LS bits of a

6: H[j] $\leftarrow b$

7: **end for**

8: **for** $i \leftarrow 0$ to $m - 1$ **do**

9: $f \leftarrow 0$

10: **for** $t \leftarrow 1$ to L **do**

11: if ($i = H[t]$) **then**

12: $f \leftarrow 1$

13: end if

14: end for

15: **if** ($f = 0$) **then**

16: $KEK \leftarrow \{K_1^j\}$

17: $S \leftarrow S$ - {Users who are descendants of nodes with key K_1^j}
18: **end if**
19: **end for**
20: **end for**
21: **Step 2:** ▷ Finds combination of keys
22: **for** each user $u_i \in S$ **do**
23: $k \leftarrow 0$
24: **for** $j \leftarrow 1$ to h **do**
25: $a \leftarrow j$ $[log_2 m]$ LS bits of UID of u_i
26: $b \leftarrow [log_2 m]$ MS bits of a
27: $k \leftarrow k \oplus K_{(h-j+1)}^b$
28: $f \leftarrow 0$
29: **for** $i \leftarrow 1$ to L **do**
30: **if** ($a = j[log_2 m]$ bits of $P[i]$) **then**
31: $f \leftarrow 1$
32: break
33: **end if**
34: **end for**
35: **if** ($f = 0$) **then**
36: break
37: **end if**
38: **end for**
39: $KEK \leftarrow KEK \cup k$
40: **end for**
41: **Step 3:** return KEK
(1) Computes $C_K = \{GK'\}_K$, $K \in KEK$
(2) Broadcasts (C_K, KID of K), $K \in KEK$

Upon receiving the broadcast message, each remaining member of the group finds the decryption key by comparing the *KID*s of its keys with the KIDs of the received messages.

It is required to change the auxiliary keys along the path where users have left. The members of the group and KDC compute auxiliary keys on their own as follows:

(1) Each member performs exclusive-OR operation of the previous auxiliary key with the new group key *GK*'.

(2) KDC performs exclusive-OR operation of the previous auxiliary key with the new group key *GK*'.

Since all the users in the group are computing auxiliary keys on their, this reduces the communication cost required to convey auxiliary keys to appropriate members of the group. Hence, KDC only conveys the new group key to the members of the group and the members will in turn compute the auxiliary keys on their own.

3.6 Storage required

We compare the amount of storage required in our scheme with the schemes proposed by Wong et al. (1998) and Chang et al. (2004). Bursty behavior scheme addressed by Zou, Ramamurthy, and Magliveras (2002) requires same amount of storage as in Wong et al. scheme (Wong et al., 1998). Table 1 depicts the amount of storage required at KDC and at each user in the group for the above mentioned three schemes. In Table 2 we have considered some specific values for the group size, N, and theoretically estimated the amount of storage required at KDC and each user.

For instance, for a group with $N = 2{,}048$ users, the height of the key tree in Chang et al. (1999) scheme is $h = [log_2 2048] = 11$. Each user in the group stores keys along the path from leaf to root. Thus, the storage required at each user is $h + 1 = 12$. KDC has to store all the keys in the key tree. In Chang et al. (1999) scheme, since only two keys are used at each level of the key tree, KDC has to store $2h + 1 = 23$ keys. For the same value of N, for a key tree with degree 3, height of the tree is $h = [log_3 N] = 7$ and for degree 4 key tree, $h = [log_4 N] = 6$. Hence, each user in degree 3 key tree stores 8 keys and in degree 4 key tree stores 7 keys. In degree 3 key tree, 3 keys are used at each level of the key tree and 4 keys are used in key tree with degree 4. Thus, KDC stores $3h + 1 = 22$ keys in degree 3 key tree. For 2,048 users key tree with degree 4 is not a complete tree. It contains only two children at level 1 and hence only two keys are required at level 1. But, in subsequent levels 4 keys are required which results in storing $2 + 4(h - 1) + 1 = 23$ keys at the KDC. We also have calculated the percentage savings in our scheme compared to Chang et al. (1999) scheme. For instance, percentage savings for a key tree with degree 3 for $N = 2{,}048$

users is $\frac{23-22}{23} * 100 \simeq 4$. It is evident from the values in Table 2 that we are reducing the amount of storage at the KDC by a factor of 4% to 12% and at the users by 30% to 38%.

3.7 Encryptions required

We compare the performance of our scheme with Li et al. scheme (Li et al., 2001). Performance analysis of Zou, Magliveras, and Ramamurthy (2002) binary key tree scheme (Zou, Ramamurthy, & Magliveras, 2002) is same as that of Li et al. (2001). Hence it suffices if we compare the performance of our scheme with Li et al. scheme. We compare the schemes with respect to number of encryptions required to convey new group key and auxiliary keys to the remaining members of the group after the leave event. We encounter worst case if the leaves are evenly distributed. In Li et al. (2001) and Zou, Ramamurthy, and Magliveras (2002) schemes it results in worst case if there is a leave from each subgroup rooted at level ($[log_2 N] - 1$) and in our scheme if there is a leave from different positions of each subgroup rooted at level ($[log_m N] - 1$). Thus, minimum number of leaves that result in worst case in both the cases is N/m. For a key tree with degree m, maximum

Table 1 Comparison of Storage Required at KDC and Each User

	Schemes		
	Wong et al. (1998) and Zou, Ramamurthy, and Magliveras (2002)	Chang et al. (1999)	Our Scheme
Storage at KDC	$2N - 1$	$2log_2 N + 1$	$min\ \{3log_3 N + 1, 4log_4 N + 1\}$
Storage at each user	$log_2 N + 1$	$log_2 N + 1$	$min\ \{log_3 N + 1, log_4 N + 1\}$

Table 2 Percentage Savings in Storage at KDC and Each User

Storage at	N	Chang et al. (1999) Scheme	Our Scheme		Percentage Savings	
			$m = 3$	$m = 4$	$m = 3$	$m = 4$
KDC	2,048	23	22	23	4	0
User	2,048	12	8	7	33	41
KDC	2,187	25	22	25	12	0
User	2,187	13	8	7	38	46
KDC	3,000	25	22	25	12	0
User	3,000	13	8	7	38	46
KDC	8,192	27	26	27	4	0
User	8,192	14	9	8	35	42

number of encryptions required to convey new group key and auxiliary keys in case of Li et al. scheme is $\frac{m(N\text{-}1)}{m\text{-}1}$. In our scheme it requires atmost $\frac{N(m\text{-}1)}{m}$ encryptions. We compute percentage savings in encryption cost in our scheme compared to Li et al. (2001) and Zou, Ramamurthy, and Magliveras (2002) schemes as $\frac{\frac{m(N\text{-}1)}{m\text{-}1} - \frac{N(m\text{-}1)}{m}}{\frac{m(N\text{-}1)}{m\text{-}1}} * 100$. For $m = 3$, this is approximately equal to 55. Thus, for a key tree with degree 3 we achieve a saving of around 55% encryption cost as compared to Li et al. (2001) scheme.

Figure 1 shows an example key tree with $N = 16$ users, u_0 through u_{15} and the degree m of the tree is 4. The binary strings 0000 through 1111 are the *UID*s of users u_0 through u_{15} respectively. The keys K_0 through K_{15} are private keys of users u_0 through u_{15} respectively (which is not shown in the figure). Here, height h of the tree is $[log_m N] = 2$. Keys K_1^1, K_1^2, K_1^3, and K_1^4 are auxiliary keys at level 1 and keys K_2^1, K_2^2, K_2^3, and K_2^4 are auxiliary keys at level 2 and GK is the group key. User u_1 for instance, stores the keys K_2^1, K_1^0 and GK which are along the path from its position till the root of the tree along with its private key K_1.

Suppose users u_2, u_3, u_8 and u_9 leave the group at the same time. Algorithm 1 considers all the leave requests and finds out encryption keys to convey new group key to the remaining members of the group. From Algorithm 1, following keys are obtained as encryption keys: since no user is leaving from the subgroups with auxiliary keys K_1^1 and K_1^3, same keys are used as encryption keys for the respective subgroups. From the subgroup with K_1^0 as the auxiliary key, users u_2 and u_3 are leaving. Hence, to convey new group key, we cannot use K_1^0. Next possibility is to look for the keys at the next lower level i.e., K_2^0 and K_2^1. Since users u_8 and u_9 are also leaving, keys K_1^0 and K_1^1 cannot be used as encryption keys. Thus, the encryption keys for u_0 and u_1 are computed as $K_2^0 \oplus K_1^0$ and $K_2^1 \oplus K_1^0$ respectively. Similarly, for users u_{10} and u_{11}, $K_2^2 \oplus K_1^2$ and $K_2^3 \oplus K_1^2$ are the respective encryption keys.

KDC broadcasts the following message:

$$\{\{GK'\}_{K_1^1},\ \{GK'\}_{K_1^3},\ \{GK'\}_{K_2^0 \oplus K_1^0},\ \{GK'\}_{K_2^1 \oplus K_1^0},\ \{GK'\}_{K_2^2 \oplus K_1^2}\ ,\ \{GK'\}_{K_2^3 \oplus K_1^2}\}$$

Now, the auxiliary keys K_1^0 and K_1^2 must be changed by both KDC and respective users. Following are the computations performed by the users to change these auxiliary keys:

Users u_0 and u_1 compute $K_1^{0\prime} = K_1^0 \oplus GK'$

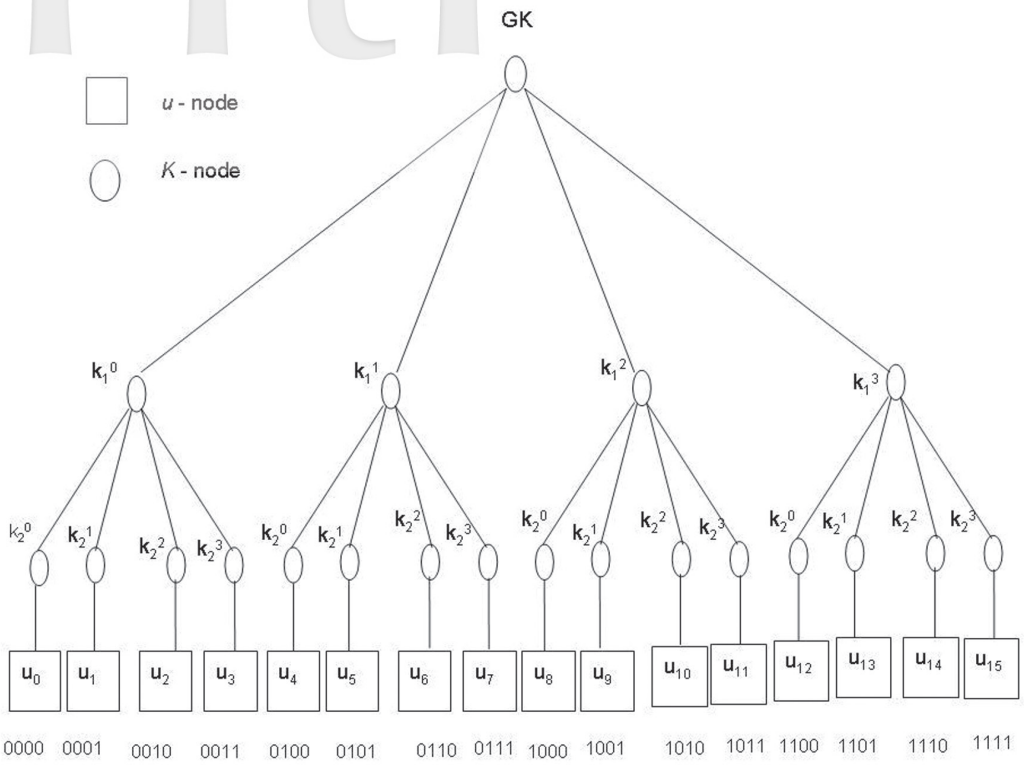

Figure 1 Key Tree Structure Showing UIDs and Keys of Users in the Group, Auxiliary Keys and Group Key

Users u_{10} and u_{11} compute $K_1^{2\prime} = K_1^2 \oplus GK'$

KDC computes $K_1^{0\prime} = K_1^0 \oplus GK'$ and $K_1^{2\prime} = K_1^2 \oplus GK'$

4. Handling bursty behavior

In this section we consider bursty behavior and analyze our scheme with respect to number of new keys generated, encryptions performed and rekey messages constructed.

4.1 Equal number of joins and leaves

We assume that initially there are N users, u_1, u_2, ..., u_N in the group. If J users join the group and L users leave the group at the same moment, where $0 \le J \le N$ and $0 \le L \le N$, then there might be a scenario in which $J = L$. In this section, we consider such a scenario and analyze with respect to the number of new keys generated, number of encryptions performed and number of rekey messages constructed.

For a group of N users, height of the key tree $h = [\log_m N]$. There are m^i nodes at

level i of the key tree, $i = 1, 2, ..., h$. It results in worst case if at least $[N/m]$ users leave the group that too each from different position of different subgroup at level $h - 1$. This scenario requires changing all the keys in the key tree. Hence for analysis purpose, we consider $L = [N/m]$. Even if more than $[N/m]$ users leave the group, the values obtained will be less than or equal to that of the values obtained in case of $[N/m]$ leaves.

Suppose L users $u_{l1}, u_{l2}, ..., u_{lL}$, $(L \leq N)$, send leave request and at the same time J users $u_{j1}, u_{j2}, ..., u_{jJ}$, send join request. Since $J = L$, joining users occupy the positions of leaving users. We first handle the leave event as in the case of cumulative member removal and then handle join event. KDC computes encryption keys required to encrypt the new group key to the remaining members of the group using Algorithm 1. Remaining users in the group find respective decryption key to decrypt the new group key by comparing the KIDs of the keys. After receiving the new group key, remaining members of the group compute the auxiliary keys on their own by performing exclusive-OR operation of the respective auxiliary keys with the new group key. For the new users, KDC encrypts the new group key and auxiliary keys using the private keys of new users and conveys the keys along with the *KID*s.

4.1.1 Number of new keys generated

For a group of N users, height of the key tree $h = [log_m N]$. Along with the group key, there are m keys at each level of the tree. Total number of keys in the key tree = $mh + 1$. Since for the worst case we have considered, all the keys must be changed, number of new keys generated = $mh + 1$.

4.1.2 Number of encryptions

We are considering worst case scenario in which at least one user leaves from each subgroup at level $h - 1$ and at least one user leaves from each position of the key tree. It requires encrypting the new group key for each remaining user in the group individually. For each new user, keys along the path from its joining point till the root are encrypted with its private key. Since $J = L = [N/m]$, there are N users in the group after handling join and leave. Thus, total number of encryptions required is N.

4.1.3 Number of rekey messages

For no two users in the group, a common rekey message is constructed. There are N users in the group after handling join and leave events. Thus, number of rekey messages required is N.

4.2 More number of joins than leaves

Suppose L users $u_{l1}, u_{l2}, ..., u_{lL}$ send leave request to leave the group and J users u_{j1},

u_{j2}, ..., u_{jJ} send join request to join the group at the same moment and $J > L$. To analyze the worst case, we are considering maximum value of L as $[N/m]$ i.e., it results in worst case if $[N/m]$ users leave the group from different positions of different subgroups at level $h - 1$. For J to be greater than L, minimum value for $J = L + 1 = [N/m] + 1$.

Among J joining users, first L users u_{j1}, u_{j2}, ..., u_{jL} occupy the positions of leaving members. We have to find either empty locations or allow the tree to grow for accommodating remaining $(J - L)$ users i.e., for users u_{jL+1}, u_{jL+2}, ..., u_{jJ}. Let the number of empty locations available in the key tree be E.

(1) If $E \geq (J - L)$, new users u_{jL+1}, u_{jL+2}, ..., u_{jJ} are inserted in the empty locations.

(2) Otherwise, if $E > 0$, first E users u_{jL+1}, u_{jL+2}, ..., u_{JL+E} are inserted in the key tree. To accommodate remaining $J - L - E$ new users, *m-ary* tree with height $h' = log_m(J - L - E)$ is constructed.

A. If $h > h'$

- the tree with height h is allowed to grow by one level in upward direction.
- the tree with height h' is allowed to grow in upward direction till h' becomes equal to $(h - 1)$.
- tree with height $h' = h - 1$ is attached as a child to the tree with height h.

B. If $h < h'$

- If the tree with height h' is complete, it is allowed to grow by one level in upward direction.
- the tree with height h is allowed to grow in upward direction till h becomes equal to $(h' - 1)$.
- tree with height $h = h' - 1$ is attached as a child to the tree with height h'.

KDC finds encryption keys using Algorithm 1, generates new group key *GK*' and broadcasts encrypted *GK*'. Remaining users in the group find decryption key by comparing the *KID*s and decrypt the new group key *GK*'. Both KDC and remaining users in the group change the auxiliary keys. For the new users, KDC encrypts the new group key and auxiliary keys using the private keys of new users and communicates the keys with their *KID*s.

4.2.1 Number of new keys generated

For a group of N users, height of the key tree $h = [log_m N]$. Along with the group key, there are m keys at each level of the key tree. Total number of keys in the key tree = $mh + 1$. Since we have considered worst case leave event, it requires changing all the keys of the

key tree. Further, to handle $J - L$ users join event, $mlog_m(J - L)$ new keys are generated. Number of new keys generated = $m(h + log_m(J - L)) + 1$.

4.2.2 Number of encryptions

We are considering worst case scenario in which at least one user leaves from each subgroup at level $h - 1$ and at least one user leaves from each position of the key tree. It requires encrypting the new group key for each remaining user in the group individually. For each new user, keys along the path from its joining point till the root are encrypted with its private key. There are $N + J - L$ users in the group after handling join and leave. Thus, total number of encryptions required is $N + J - L$.

4.2.3 Number of rekey messages

For no two users in the group, a common rekey message is constructed. There are $N + J - L$ users in the group after handling join and leave events. Thus, number of rekey messages is $N + J - L$.

4.3 Number of joins less than number of leaves

Suppose L users u_{l1}, u_{l2}, ..., u_{lL} send leave request to leave the group and J users u_{j1}, u_{j2}, ..., u_{jJ} send join request at the same moment and $J < L$. Since J is less than L, sometimes the key tree may shrink in height. We encounter worst case scenario if $L = [N/m]$ and each user leaves from different position of different subgroup at level $h - 1$. Since we are considering the case for $J < L$, the value of J can range from 0 to $L - 1$. New J users occupy the positions of first J leaving users.

KDC finds encryption keys using Algorithm 1, generates new group key *GK*' and broadcasts encrypted *GK*' along with the *KID*s of encryption keys. Remaining users in the group find decryption key by comparing the *KID*s and decrypt the new group key *GK*'. Both KDC and remaining users in the group change the auxiliary keys. For the new users, KDC encrypts the new group key and auxiliary keys using the private keys of new users and sends the keys along with the *KID*s.

4.3.1 Number of new keys generated

For a group of N users, height of the key tree $h = [log_m N]$. Along with the group key, there are m keys at each level of the key tree. Total number of keys in the key tree = $mh + 1$. Since we have considered worst case leave event, it requires changing all the keys in the key tree. Since joining users occupy the positions of some of the leaving users, join event does not require generation of new keys. Number of new keys generated = $mh + 1$.

4.3.2 Number of encryptions

We are considering worst case scenario in which at least one user leaves from each

subgroup at level $h - 1$ and at least one user leaves from each position of the subgroup. It requires encrypting the new group key for each remaining user in the group individually. For each new user, keys along the path from its joining point till the root are encrypted with its private key. There are $N + J - L$ users in the group after handling join and leave. Thus, total number of encryptions required is $N + J - L$.

4.3.3 Number of rekey messages

For no two users in the group, a common rekey message is constructed. There are $N + J - L$ users in the group after handling join and leave events. Thus, number of rekey messages is $N + J - L$.

4.4 Performance comparison

We compare the performance of our scheme with the binary key tree schemes proposed by Li et al. (2001) and Zou, Ramamurthy, and Magliveras (2002) with respect to number of new keys generated, encryptions performed and rekey messages constructed. Table 3 gives the comparison. It is observed that the number of new keys generated in the schemes of Li et al. (2001) and Zou, Magliveras, and Ramamurthy (2002) is proportional to N and in our scheme it is proportional to logarithmic in the group size. Number of encryptions performed in Li et al. (2001) and Zou, Magliveras, and Ramamurthy (2002) schemes is in terms of twice the group size, whereas in our scheme it is in terms of only the group size. Thus, number of encryptions performed and new keys generated are less as compared to the ones required in Li et al. (2001) scheme and Zou, Magliveras, and Ramamurthy (2002) scheme. Reduction in number of key generation and encryptions reduces the computation cost at the KDC. Thus, our scheme outperforms the scheme of Zou, Ramamurthy, and Magliveras (2002) in both storage and computation costs.

5. Implications for practice

We find application of this SGC model in pay-per-view, periodical electronic information distribution, on-line journal subscription and so on. In pay-per-view application, users usually subscribe for specific events which are quite interesting for them. Hence, the telecast should be made available only for the subscribed period. Usually, many users subscribe for those events which are popular and this results in many users joining and leaving the event at the same time.

In periodical electronic information distribution and on-line journal subscription, many users subscribe for a particular duration, i.e., usually for one year or two years period. It results in many users subscribing at the beginning of the year (usually during the month of January) and un-subscribing at the end of the year. We may also encounter a

Table 3 Number of New Keys, Encryptions, and Rekey Messages Required for Different Cases

	New Keys		Encryptions		Rekey messages	
	Li et al. (2001) & Zou, Magliveras, and Ramamurthy (2002) schemes	Our Scheme	Li et al. (2001) & Zou, Magliveras, and Ramamurthy (2002) schemes	Our Scheme	Li et al. (2001) & Zou, Magliveras, and Ramamurthy (2002) schemes	Our Scheme
$J = L$	$N + J - 1$	$mlog_m N + 1$	$2(N - 1)$	N	N	N
$J > L$	$N + L - 2$	$m(logmN + logm(J - L)) + 1$	$2N - 4 + 4(J - L)$	$N + J - L$	$N + J - L$	$N + J - L$
$J < L$	$N + J - 1$	$mlog_m N + 1$	$2N + J - L - 2$	$N + J - L$	$N + J - L$	$N + J - L$

situation in which when many users unsubscribe, other users may subscribe for the event resulting in bursty behavior.

In the above mentioned scenarios, if the events are considered as secure groups, *m-ary* tree is constructed with subscribing users as members of the group and these users can be provided with the keys along the path. Users can have access to the application only if proper key is entered. Once new users subscribe and existing users unsubscribe, new keys can be distributed efficiently using our scheme. Thus we find application of our scheme in day-today scenarios.

6. Conclusion

We proposed a scheme which handles both cumulative member removal and bursty behavior in a secure group. We used *m-ary* key tree with only *m* keys at each level of the tree. We showed that in our cumulative member removal scheme the encryption cost is reduced by about 55% compared to the scheme proposed by Li et al. (2001). If 4-*ary* tree is used, storage required at KDC is same as the Chang et al. (1999) scheme, but at the users it is reduced. If 3-*ary* tree is used, it reduces the storage at both KDC and users. We showed that the savings in storage at KDC ranges from 4% to 12% and with users it ranges from 30% to 38%.

We considered different bursty behavior scenarios and analyzed the scheme for worst cases. In each scenario we computed number of new keys generated, number of encryptions performed and number of rekey messages constructed. We compared our proposed scheme with the LKH (Logical Key Hierarchy) scheme and showed that our scheme performs better in terms of cost of encryption and cost of key generation, thereby reducing the computation cost overhead at the KDC.

References

Amir, Y., Danilov, C., Miskin-Amir, M., Schultz, J., & Stanton, J. (2004) 'The spread toolkit: architecture and performance', Technical report, Johns Hopkins University, Baltimore, MD.

Amir, Y., Kim, Y., Nita-Rotaru, C., Schultz, J.L., Stanton, J., & Tsudik, G. (2004) 'Secure group communication using Robust contributory key agreement', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 5, pp. 468-480.

Blundo, C., De Santis, A., Herzberg, A., Kutten, S., Vaccaro, U., & Yung, M. (1993) 'Perfectly secure key distribution for dynamic conferences', *Lecture Notes in Computer Science*, Vol. 740, pp. 471-486.

Burmester, M. & Desmedt, Y. (1995) 'A secure and efficient conference key distribution system', *Lecture Notes in Computer Science*, Vol. 950, pp. 275-286.

Burmester, M. & Desmedt, Y. (1997) 'Efficient and secure conference-key distribution', *Lecture Notes in Computer Science*, Vol. 1189, pp. 119-129.

Caronni, G., Waldvogel, K., Sun, D., & Platter, B. (1998) 'Efficient security for large and dynamic multicast groups', *Proceedings of the Seventh IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises* (*WET ICE'98*), Palo Alto, CA, June 17-19, pp. 376-383.

Chang, I., Engel, R., Kandlur, D., Pendarakis, D., & Saha, D. (1999) 'Key management for secure Internet multicast using Boolean function minimization technique', *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies* (*INFOCOM'99*), New York, March 21-25, pp. 689-698.

Diffie, W. & Hellman, M. (1976) 'New directions in cryptography', *IEEE Transactions on Information Theory*, Vol. 22, No. 6, pp. 644-654.

Fiat, A. & Naor, M. (1993) 'Broadcast encryption', *Proceedings of the Third Annual International Cryptology Conference* (*CRYPTO'93*), Santa Barbara, CA, August 22-26, pp. 480-491.

Lee, P.P.C., Lui, J.C.S., & Yau, D.K.Y. (2002) 'Distributed collaborative key agreement protocols for dynamic peer groups', *Proceedings of the Tenth IEEE International Conference on Network Protocols* (*ICNP'02*), Hong Kong, China, November 12-15.

Li, X.S., Yang, Y.R., Gouda, M.G., & Lam, S.S. (2001) 'Batch rekeying for secure group communications', *Proceedings of the Tenth International World Wide Web Conference* (*WWW'01*), Hong Kong, China, May 1-5, pp. 525-534.

Lin, J.C., Lai, F., & Lee, H.C. (2005) 'Efficient group key management protocol with one-way key derivation', *Proceedings of the IEEE Conference on Local Computer Networks Thirtieth Anniversary* (*LCN'05*), Taipei, Taiwan, November 17, pp. 336-343.

Mittra, S. (1997) 'Iolus: a framework for scalable secure multicasting', *ACM SIGCOMM Computer Communication Review*, Vol 27, No. 4, pp. 277-288.

Perrig, A. (1999) 'Efficient collaborative key management protocols for secure autonomous group communication', paper presented at the International Workshop on Cryptographic Techniques and E-commerce (CrypTEC'99), Hong Kong, China, July 5-8.

Poornima, A.S. & Aparna, R., & Amberker, B.B. (2007a) 'Key management for cumulative member removal in secure group communication', paper presented at the International Conference on Embedded Systems, Mobile Communication and Computing, Bangalore, India, August 3-5.

Poornima, A.S., Aparna, R., & Amberker, B.B. (2007b) 'Storage and rekeying cost for cumulative member removal in secure group communication', *International Journal of Computer Science and Network Security*, Vol. 7, No. 9, pp. 212-220.

Rafaeli, S. & Hutchison, D. (2003) 'A survey of key management for secure group communication', *ACM Computing Surveys*, Vol. 35, No. 3, pp. 309-329.

Sherman, A.T. & McGrew, D.A. (2003) 'Key establishment in large dynamic groups using one-way function trees', *IEEE Transactions on Software Engineering*, Vol. 29, No. 5, pp. 444-458.

Waldvogel, M., Caronni, G., Sun, D., Weiler, N., & Platter, B. (1999) 'The Versakey framework: versatile group key management', *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 9, pp. 1614-1631.

Wallner, D., Harder, E., & Agee, R. (1999) 'Key management for multicast: issues and architectures', IETF RFC 2627, The Internet Society, Washington, DC.

Wong, C.K., Gouda, M., & Lam, S.S. (1998) 'Secure group communications using key graphs',

Proceedings of the ACM SIGCOMM'98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Vancouver, Canada, September 2-4, pp. 68-79.

Wong, C.K. & Lam, S.S. (2000) 'Keystone: a group key management service', paper presented at the International Conference on Telecommunications (ICT), Acapulco, Mexico, May 20-25.

Zou, X., Magliveras, S., & Ramamurthy, B. (2002) 'A dynamic conference scheme extension with efficient bursty operation', *Congressus Numerantium*, Vol. 158. pp. 83-92.

Zou, X., Ramamurthy, B., & Magliveras, S. (2002) 'Efficient key management for secure group communications with bursty behavior', *Proceedings of the IASTEC International Conference on Communications, Internet, and Information Technology* (*CIIT 2002*), St. Thomas, USVI, November 18-20, pp. 148-153.

About the authors

R. Aparna obtained her Ph.D. from Visvesvaraya Technological University, Belgaum, Karnataka, India. She is presently working as an Associate Professor in the Department of Computer Science and Engineering, Siddaganga Institute of Technology, Tumkur, Karnataka, India. Her research areas include cryptography and network security, security in sensor networks and database security.

B.B. Amberker is a Professor with Department of Computer Science and Engineering, National Institute of Technology, Warangal, Andhra Pradesh, India. He received his Ph.D. from the Department of Computer Science and Automation, Indian Institute of Science (IISc), Bangalore, India. His research areas include algorithmic cryptography, number theoretic algorithms, digital water marking and security in sensor networks.