Journal of Computer Science 2 (6): 542-549, 2006
ISSN 1549-3636

An Effective Key Management Approach to Differential Access Control in Dynamic Environments

Yogesh Karandikar, Xukai Zou and Yuanshun Dai
Department of Computer and Information Science,
Purdue University School of Science at Indianapolis, Indiana 46202, USA

Abstract: Applications like e-newspaper or interactive online gaming have more than one resource and a large number of users. There is a many-to-many relationship between users and resources; each user can access multiple resources and multiple users can access each resource. The resources are independent and each resource needs to be encrypted by a different Resource Encryption Key (*REK*). Each *REK* needs to be distributed to all subscribers of the resource and each subscriber must get all the *REKs* he/she subscribes to. Also this environment is very dynamic in terms of subscription changes by users and resource changes by service providers. We term this as the problem of key management for Differential Access Control (*DIF-AC*) in dynamic environments. Conventional ways of access control are not sufficient for this problem of DIF-AC key management. In this study we propose a novel approach of keys management to enforce DIF-AC in highly dynamic environments, based on Secure Group Communication framework.

Key words: Access control, secure group communication, dynamic conferencing, key management

INTRODUCTION

Group communication can be defined as a process where a group of n users can communicate with each other. Improvements in IP multicast have expanded the horizons of the group communication oriented applications. Applications like video conferencing, pay-per-view broadcasts, e-newspaper are representatives of the growing trend. Multicast networks are built in such a way that anyone joining a multicast group can have access to the group communication. As a result, the entire multicast based applications face the problem of security of data over the multicast network.

The problem of securing group communications is well studied. A primary method of limiting access to the information is through encryption and selective distribution of encryption key[1]. Various centralized[2,3], decentralized[4] and distributed[5,6] key management schemes are proposed in literature. Access control in all these schemes depends upon possession of the group key, which is termed as the Traffic Encryption Key (*TEK*). Users who have the *TEK* can access the group communication.

The field of broadcasting has existed for a long time. The digital cable television provided to us via set-top boxes is the best example of data broadcasting. Cable television companies provide option of choosing various channels and they tune the set-top boxes for each user accordingly. The problem of broadcast encryption is well studied[7-10]. The limiting factor in broadcast encryption of digital Cable services is the capability of the set-top boxes in terms of memory and processing power.

In this study we are focusing on applications like e-newspaper, pay-per-view broadcasts over Internet, newsgroups, real-time email lists, interactive online gaming. In applications like these there is one service provider, which can be considered as the *Central Controller*, who provides various resources like various sections of an e-newspaper. There are a large number of users registered with the *Central Controller*. Typically the number of users is many times the number of resources, as an example consider any newspaper, the number of resources (sections) offered are like say 30 whereas the number of users is in millions.

Not all million users are interested in all the sections offered. If we convert our newspaper into e-newspaper, then it is possible to give a customized copy of the paper to every user. By customized we mean, we can give only those sections to a user that he/she wants and is willing to pay for. This is like the pay-per-view cable idea where you pay only for those channels which you want to see, you do not get to see all available channels unless you are willing to pay for them and at the same time you are not charged for channels that you never watch. The pay-per-view idea can also be used for multicasting of various events in the form of audio and/or video data over the Internet. We can have many types and qualities of multicasts and users can have a choice about the event and quality. Suppose there are m resources on offer to n users,

Corresponding Author: Yogesh Karandikar, Department of Computer and Information Science, Purdue University School of Science at Indianapolis, Indiana 46202, USA

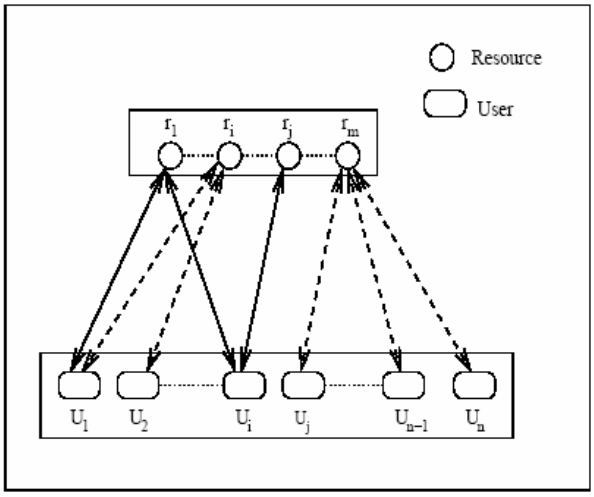

Fig. 1: Relation between m resources and n users

Fig. 1 shows the relation between the resources and users.

In the applications discussed above we can not assume a hierarchical relation between various sections of an e-newspaper or various quality multicasts. If a user subscribes for political section of the e-newspaper, there is no way to assume that he/she would subscribe to sports section. It is not correct to force any hierarchical relation between resources. In this study we propose a scheme based on the general assumption that the resources are independent with no hierarchical relation between them. Even if some applications have some inherent hierarchy between resources, it can be considered as a special case and our scheme is generic enough to be easily extended for those applications.

As each resource is independent, it must be encrypted by a unique key[11]. Thus all the m resources must be encrypted by one unique key each, giving rise to m Resource Encryption Keys (*REKs*). We have one set of users associated with each resource, giving rise to m sets, which we term as m Resource Lists (*RLs*). The users in these *RLs* can overlap as different users subscribe to different combinations of resources. We term the type of access control required for this complex scenario as Differential Access Control (*DIF-AC*). The *DIF-AC* comes with an even bigger problem and that is of distributing the different *REKs* to various subsets of users (*RLs*), because if a user subscribes to j resources, he/she is a member of j *RLs* and must get j *REK*s. Also in a truly dynamic environment, users can add or drop subscriptions and *Central Controller* can add or drop services.

In order to find a generic solution we need to consider the fact that some applications can have users communicating with each other like in newsgroups or online gaming. The service provider may need to communicate with all or a part of the users like promoting new services to all existing users or sending some confidential information only to registered users or giving some special discounts only to some users. In either case a secure communication medium between users is required. The most efficient and scalable way to distribute keys is considered to be construction of Key Distribution Trees[2]. Hence, a good solution to solve the problem of key distribution for *DIF-AC* in dynamic environments should be based on key tree scheme like 'The efficient key tree scheme for Bursty Operation"[3].

A lot of work has been done in the area of access control. Access control policies and principles are well described in[12,13]. Also there are a number of schemes for Hierarchical Access Control[14-16]. The dual encryption protocol for *SGC*[4] assumes one sender and multiple receivers. It should be noted that there is not much work done for *DIF-AC* which arises due to the many-to-many relationship between users and resources.

The only literature that deals with the problem of key management for *DIF-AC* scenario is the scheme proposed by Sun *et al.*[17]. They assume m resources and denote list of resources as $\{r_1, r_2, \ldots, r_m\}$. With each resource a Data Group (*DG*) is associated, thus they have m *DG*s as $\{D_1, D_2, \ldots, D_m\}$. They also have service groups *SG*s as $\{S_1, S_2, \ldots, S_n\}$ where $n = 2^m - 1$. Basically each *SG* represents a unique possible combination of resources that can be subscribed by any user. Members in different *DG*s can overlap, but members in different *SG*s can not overlap. Each resource is to be encryped by a separate key[18] i.e. one key is associated with each *DG*. For example if S2 represents a *SG* with combination of resources $\{r_1, r_3, r_4\}$ then all users in S_2 must have keys of $\{D_1, D_3, D_4\}$. Each D_i contains a list of members that subscribe to the resource r_i and all those users must get the key of D_i. Thus if resource r_1 is part of S_1, S_2, S_{10} all members in those *SG*s must have key of D_1.

Figure 2 shows the steps followed in the scheme. In the first step, one key tree is created for each *SG* for a total of $n = 2^m - 1$ trees. These trees have users at leaf nodes and *SG keys* at their roots. Then in the second step m trees are created, one for each *DG*. In the *DG* trees *SG keys* are at leaf nodes and the roots of *DG* trees are the *DG keys*. Then all these are integrated to get a Key Graph by forcing hierarchy.

This scheme is inefficient and far from practical. It is required to create m trees for *DG*s and $2^m - 1$ trees for *SG*s. The procedure to create trees for *DG* is complicated as each resource can be a part of 2^{m-1} *SG*s and each *DG* tree must contain all the *SG*s as its leaf nodes. Procedure to merge the trees to get one integrated Key Graph is quite complex and seems impossible to automate. If m is a large number say 10, we will have $2^{10} - 1 = 1023$ *SG* trees and 10 *DG* trees. Each resource will be a part of ($2^9 = 512$) *SG*s, so each *DG* tree requires to have 512 leaf nodes. Creating all these trees and merging them in one integrated tree graph are going to be really complex and seems far from practical. As a matter of fact, even for the toy example with 2 resources shown in Fig. 2 the procedure to merge trees is not clear.

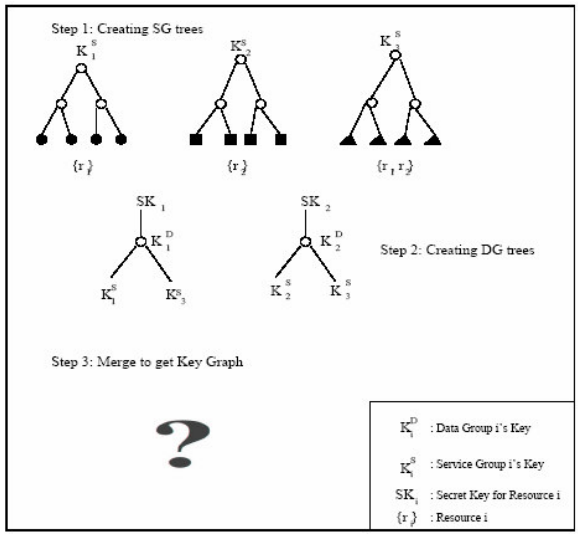

Fig. 2: Scalable Hierarchical Access Control Scheme for *SGC*, *Yan Sun et.al.*

To add to this complexity just imagine what happens if the service provider wants to add or remove a resource. For example, let there be 5 resources. The number of *SG*s will be 31 and hence the number of *SG* trees is 31 and number of *DG* trees is 5. Each *DG* tree will have 16 leaf nodes. Now, if the service provider wants to add 2 resources, the entire graph needs to be redone. The reason is that the number of *SG*s with 7 resources will be 128 and there are combinations that were not present with 5 resources. Or consider a case where a resource is revoked. It is equally worse to deal with. Also since *SG*s present all possible combinations of resources, not all *SG*s will have some users in them all the times. Even if there are no users present in an SG, that SG must still be present in the DG tree. Overall this scheme is too complex and far from practical.

Our proposed Scheme is very generic in the sense that it does not force any resource or user hierarchies that are not present. Our scheme provides a secure communication channel between users. Our scheme will have only m + 1 key trees at maximum making it efficient and will be completely dynamic in terms of users and resources. In the next section we describe the key tree based *SGC* and *SDC* schemes. Although we are using the binary key tree based scheme to illustrate our scheme, it is not a restriction. Our scheme can be deployed with any n-ary key tree as well.

Key tree based secure dynamic conferencing: The Tree based key management scheme (called Key Tree) is a well-known, efficient and scalable solution for the *SGC* key management problem[2,3,19, 20,]. Based on the Key Tree Scheme and the *SDC* scheme we propose a new Scheme for *DIF-AC* key management in dynamic environments. We make use of "Efficient Key Tree based Scheme with Bursty operations"[3] for finding secure communication channels (i.e., the internal nodes) covering users and distributing the *REKs*. The key tree based *SDC* scheme[21], will be used to decide *REKs*. The Key Tree Scheme and the *SDC* scheme are explained in this section.

Key tree scheme: There is a centralized group controller (*GC*), which maintains the group and manages a virtual tree (Fig. 3).

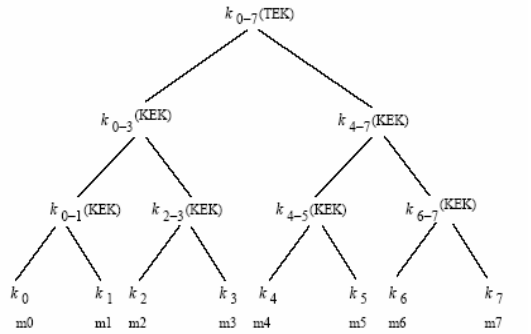

Fig. 3: A typical key tree with members at leaf nodes

The members of the group are placed at leaf nodes of the tree. Every node in the tree is associated with a key. Every member is assigned the keys along the path from its leaf to the root. The key at the root, called as the traffic encryption key (*TEK*), is shared by all the members and is used for encrypting all group messages. All other keys are called key encryption keys (*KEK*s). When a member joins or leaves, all the keys from the root to the parent of the member will be changed by the *GC* (from bottom up). Every changed key will be encrypted with its children's keys and broadcast to all members. When the key tree is a binary tree, the number of keys which need to be changed for a join or a leave is O(log(n)) where n is the number of members in the group.

In the domain considered for this study, this kind of scheme is ideal as there is one Central Service Provider with many registered users associated. The Service Provider can be seen as the Group Controller and registered users as the members of the Key Tree. In applications like e-newspaper or pay-per-view broadcasts, the users may not want to communicate with each other but the Key Tree based scheme provides a framework for sending all the material targeted for all registered users to be encrypted with *TEK* and broadcast. So if the service provider wants to send some special offers or some news only to the registered users, the provider can encrypt that offer message by *TEK* and broadcast it. In applications like online-gaming or newsgroup users need to communicate with each other and hence a *SGC* framework is required.

Secure dynamic conferencing: If a subset of members, of the group, want to have a conference

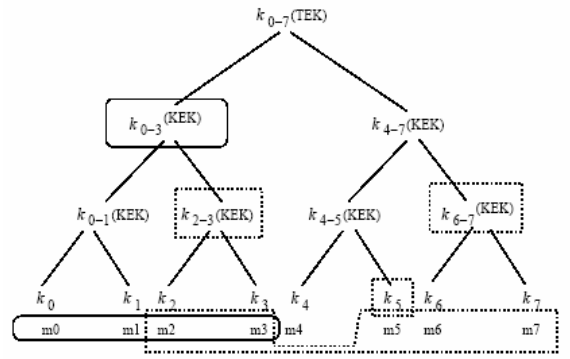

Fig. 4: Conference members are covered under nodes

among them, a conference key needs to be distributed among the conference members securely and efficiently. We have a simple and efficient scheme to accomplish conference key distribution[21]. If the conference members happen to *exactly* be covered under a single node in the key tree, the key on the node will be the conference key.

For example, in Fig. 4, m0 initiates a conference containing $\{m_0, m_1, m_2, m_3\}$ which are exactly covered under node k_{0-3}. So, k_{0-3} will be used as their conference key. This is one case. In the second case, two steps will be performed: (1) The conference initiator determines (the indices of) the keys in the key tree which *exactly* cover the conference members, randomly selects a new conference key CK, encrypts CK with its leaf key, and sends the encrypted CK along with the key indices to the *GC*; (2) the *GC* decrypts CK, picks up the keys corresponding to the indices, encrypts CK with each of these keys, and broadcasts to the group.

Let us see an example from Fig. 4 again. Suppose m2 initiates a conference containing $\{m_2, m_3, m_5, m_6, m_{27}\}$ (dotted boxes in the figure). The nodes which *exactly* cover these members are k_{2-3}, k_5; k_{6-7}. m_2 will encrypt a randomly selected new CK with its leaf key k_2 and sends it along the key indices ({2-3, 5, 6-7}) to the *GC*. The *GC* decrypts CK using k_2, encrypts CK with k_{2-3}, k_5, and k_{26-7} respectively, and broadcasts the encrypted CKs. Thus m_3, m_5, m_6 and m_7 can decrypt the CK after receiving the broadcast.

We modify and extend the *SDC* idea to use it in the domain of the paper. We view the members of a *RL* as the members in a conference. Thus for m *RLs* we have m conferences. We assume that the *GC* initiates all the conferences and finds CK and distributes to the conference members, making use of the established Key Tree. In case the number of keys that cover a conference is large (worst case scenario where members of conference are dispersed) the *GC* can create a tree with cover keys at the leaf nodes and CK at the root for that conference. Based on this *SGC* framework, we will propose our new efficient key management scheme for *DIF-AC* in dynamic environments.

Differential access control scheme: In the domain considered, the factors to be considered are 1) A central Service provider offers a number of resources; 2) There are large number of users, who are registered with the service provider; 3) There should be a secure communication medium between users to make the scheme generic; 4) Each user can subscribe to any number of resources from none to all. Also each user can subscribe to any combination of resources from all the available resources; 5) There may not be any hierarchical relation between resources and in order to maintain many-to-many relationship no hierarchy can be forced; and 6) The most important point is that it is completely dynamic. A user can switch from subscription of one resource to another if he/she is not violating access control policies. The other dynamic part is the Central Service Provider can add or revoke a resource at any point of time without affecting the entire key distribution scheme.

System description and initialization: Let there be m resources in the system $\{r_1, r_2, \ldots, r_m\}$. We will have a resource list associated with each resource, so there will be m resource lists as $\{RL_1, RL_2, \ldots, RL_m\}$. Let there be n users registered with the *GC*. The *GC* will create and manage a Key Tree for these n users and efficient algorithms proposed in[3] will be implemented to allow bursty join and/or leaves. The nodes of the Key Tree will be numbered as follows: The root is numbered 0. For the rest of the nodes, node number of left child of any node is obtained by left shifting the parent node number by 1-bit and adding one to it. Similarly node number of right child is obtained by adding one to parent node number and then left shifting by one bit. For example if parent node number is 2, then its left child is 2 << 1 + 1 i.e. 5 and right child is (2+1) << 1 i.e. 6 (Fig. 5). Due to this indexing scheme, indices of the left children nodes are odd and indices of the right children are even. The *GC* will maintain the m resource lists (*RL*s) for m resources. Each *RL* will have a list of members who have subscribed for that resource. The *RL*s will be initialized to be empty at the start when system is being set up. Every subscription request will be kept in an appropriate *RL*. Thus m *RL*s will be created in a time interval and maintained by the *GC*. These m *RLs* correspond to m conferences.

At the end of time interval the *GC* will run the algorithm to determine key indices of a conference (Fig. 6) for each *RL*. This algorithm starts by sorting the IDs of users in a conference (*RL*). In the first pass, pairs of users covered under the same node are identified and sorted in an array. In the next pass, the algorithm checks if two pairs are covered under one node. This continues until all the cover keys are identified.

To identify if two users are covered under one node, the indexing scheme is helpful. If the first user's

index is even, that means that that user will not share a key with any other user since he/she is on a right branch. Thus that user's index is put on the cover key list and that users key is one of the cover keys. If the user's index is odd, next user's index is checked to see if they share the key. After the first pass, the same operations are done but on pairs of users identified in first pass. Next passes are similar and they either identify pairs that share node keys or are separate and hence cover keys. We can apply some more optimization techniques here like if k *RL*s are identical then the same *REK* can be used for all k *RL*s.

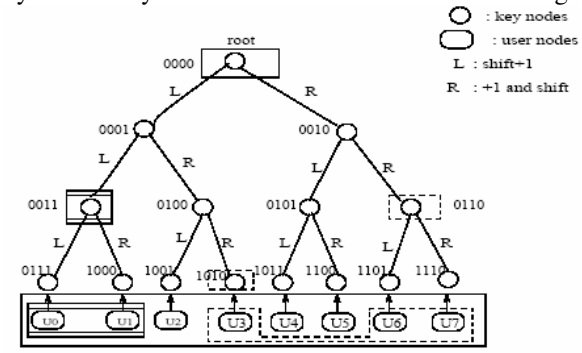

Fig 5: Binary indexing of keys

```
//r: RL size; i_0, i_1, i_{r-1}: members' IDs list
//Resource List is sorted in ascending order of member IDs
While (current member ID list is not empty) {
    if (Current_ID is even) {
      Store Current_ID (which is one key index needed)
      Remove the Current_ID from current member ID list
      Move Current_ID to the next ID
    } else {
      if (Next_ID equals Current_ID +1) {
        Find parent_ID of Current_ID
        ReplaceCurrent_ID by Parent_ID
        Remove Next_ID
        Move Current_ID to the next ID
      } else{
        Store Current_ID (which is one key index needed)
        Remove the Current_ID from current member ID list
        Move Current_ID to the next ID
      }
    }
}
```

Fig. 6: An efficient recursive algorithm for determining the key indices for any *RL*

For example, consider an e-newspaper offering 3 sections: sports, politics and stocks. Assume 8 members are registered with the *GC* and the *GC* maintains a Key Tree as in Fig. 5. If all the members subscribe for sports section (resource 1) then they are covered under the root key, hence the *REK* for sports is the root. If members m_0 and m_1 subscribe to resource 2, i.e., politics, they will be put in RL_2. Now, members in RL_2 are also covered by a single key (K_3) which will be the *REK* for politics. Finally, if members m_3,m_6 and m_7 subscribe to stocks (resource 3), the *GC* will find the cover keys K_{10} and K6. The *GC* will generate a *REK* for resource 3 and send it using the cover keys. If the number of members in a *RL* is really large, which results in a large number of cover keys, a tree can be constructed to distribute *REK*. The tree constructed for distributing *REK* can be similar to the central Key Tree in terms of structure and algorithms and hence will be efficient and scalable. In the worst case we will have m trees for m resources and 1 central Key Tree. The decision to create a tree for *REK* distribution is based on the number of cover keys and we can have a threshold value for that.

The basic idea behind our scheme is the fact that User subscriptions to a resource can be viewed as a special case of SDC. Although *SDC* was proposed for a different domain, it can be used efficiently for the domain of the paper. The many-to-many relation between users and resources makes it really complex and discourages use of forced hierarchies. Allowing users to change subscriptions and the *GC* to change resources makes the Environment very dynamic. Once the central Key Tree and all the m Resource Lists (*RL*s) are set-up, the *GC* just has to maintain user joins and leaves and subscription changes. The next subsection describes the dynamic operation handling.

System Dynamics and Maintenance: We will assume that the *GC* will have a pre-defined time interval and all updates will be done only at the end of time interval. The time interval chosen should be large enough to avoid frequent updates and small enough to avoid loss of revenue. Choosing the time interval is application dependent. We will first deal with dynamics of subscription to resources. There are 3 possible cases:

* A member subscribes to one or more new resources to which he has not already subscribed.
* A member un-subscribes from one or more resources already subscribed.
* A member changes from subscribed resources.

To handle the subscription dynamics, the *GC* will have temporary *RL*s called as *TRL*s which will be initialized to be copies of corresponding *RL*s at the start of a time interval. These TRLs will be mapped to RLs at the end of time interval. To handle the first case, whenever the *GC* gets a *subscription* request from a registered member, the *GC* puts that member in the TRLs for the requested resources. Similarly for *un-subscription* requests, the *GC* just has to delete the member from the TRLs of the resources requested. To change subscription, the *GC* has to simply delete member from the TRLs of the resources where he/she wants to unsubscribe and put him/her in the TRLs of the resources requested. For example, if a member sends a request to the *GC* for change of subscription from $\{r_i, r_j\}$ to $\{r_k, r_l\}$. The *GC* will delete that member from $\{TRL_i, TRL_j\}$ and add him/her to $\{TRL_k, TRL_l\}$.

At the end of time interval the *GC* will run the efficient algorithm to find *REK* (Fig. 6) for each *TRL* where there was a change. One dirty bit per *RL* will help us detect the lists that were modified. All the changed TRLs will be mapped to corresponding RLs. The new cover keys will not differ from the original cover keys as most of the users will be same. This can be seen as a special case of some members leaving and joining from an ordinary Key Tree. The mapping process can be executed with the efficient algorithms for Key tree updates discussed in Section 3. In the maintenance operations described it is assumed all the legal and technical formalities related to subscription, un-subscription and changes are taken care of.

Now let us describe resource dynamics. There are also only three possible cases:

* Adding one or more resources, like an e-newspaper, adding a soccer news section.
* Revoking a resource, like a broadcaster, revoking a 56Kbps quality broadcast.
* Changing a resource, like a broadcaster changing the 56Kbps quality broadcast to 128kbps.

Our scheme makes it really simple to incorporate these resource dynamics. To add a resource, the *GC* just has to add a new Resource List *RL* corresponding to that resource to the existing *RL*s. To revoke a resource, the *GC* just deletes the corresponding *RL*. To change a resource, the *GC* does not have to do anything at all, as RLs are just like interfaces in the Object Oriented world and it is fine to change implementation as far as interface is same. Thus if a broadcaster wants to change a 28Kbps quality resource to 56Kbps resource, he/she is free to do so without affecting the subscribers and the system, or if an e-newspaper wants to replace a section like politics by world politics, only the resource changes and not any other part of the system.

In case a user wants to un-subscribe from all resources and leave the system all together, he/she can send a leave request to the *GC*. The *GC* will delete that leaving member from the *RL*s that he was part of and remove him/her from the central Key Tree updating all affected keys. This is like a normal leave from any Key Tree scheme with extra operation of deleting from *RL*s.

DISCUSSION

We will discuss various performance and security issues[11] of our scheme in this section. We will discuss the best, worst, and average case scenarios of finding the Resource Encryption Keys (*REK*) for each resource, followed by issues of scalability, number of keys and dynamics in terms of user join/leave and resource addition and revocation.

The *Best Case* of finding *REK*s is when all the users in a *RL* are covered under one key. Then that key from the efficient key tree for *SGC* becomes the *REK*. In the best case it is not required to construct a tree for distributing the *REK*. The *Worst Case* of finding *REK* is when we have n=2 members in a *RL* in such a way that there is no shared key between any two members.

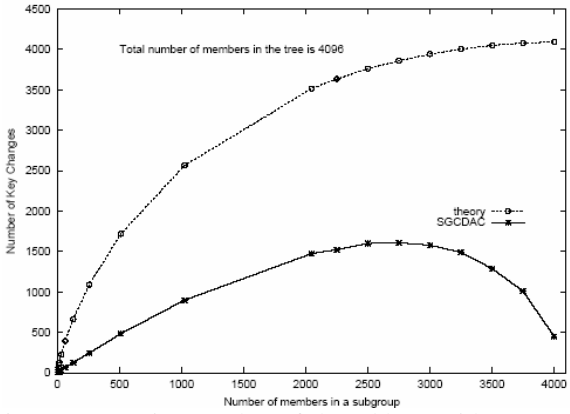

Fig 7: Comparing number of shared keys with N_{avg}

In this case, the algorithm to find key cover returns n=2 keys. In this case, it may be more efficient to construct a separate key tree with all the cover keys at leaf nodes and the *REK* at the root. In the average case, we will have more than one and less than N_{avg}=2 keys. Where, N_{avg} is the average number of keys to be changed in the batch rekeying scheme described in Chapter 3 of the book *SGC* over Data Networks[22].

$$N_{avg} = \sum_{l=0}^{h-1} 2^l \left(1 - \frac{\binom{\frac{n}{2^l}}{m}}{\binom{n}{m}} \right).$$

where n is the total number of members in a key tree, h the height of the tree, and m the number of the changed members. N_{avg} gives the total number of keys to be changed from the root to leaves of changed members. In our case, we just need to find the keys shared by members in a RL, hence the average number of keys is bound to be less than N_{avg}/2. We are not concerned with all keys till the root.

We ran an experiment with the height h of a tree as 12 and the total number n of members as 4096. We changed the number (m) of members in subset from 8 to 4000. We calculated N_{avg} using the formula described above and the shared keys using the algorithm described in Fig. 6. The results are plotted in Fig. 7. As it can be seen from the figure, the average number of shared keys is far less than the expected value of N_{avg} /2. Let us discuss the number of trees created. In the worst case, it will be m + 1, which is equal to the number of resources +1 for the central key Tree, which is lot better than the number of trees created with the scheme proposed by

Sun et.al[17]. In the average case the number of trees will be less than m + 1, as some *REK*s will be either from a central key tree or without a need to create a tree.

We can have a threshold value for the number of cover keys and we can construct a key tree for distributing *REK* if the number of cover keys exceeds the threshold. Thus our scheme is far better in terms of the number of trees constructed than existing schemes.

In terms of the number of keys, all the members will have $\log_2 n$ keys for the central Key Tree. Besides that all members will have the *REKs* for the resources they subscribe. For example, if a member subscribes for i resources, he/she will have i *REKs*. In case some *REK*s are distributed by creating trees, all members subscribing to those resources will have keys from *REK* distribution trees. As we showed earlier, the worst case number of leaf nodes of a *REK* distribution tree is n/2. Hence, if a resource has a key tree associated with it, in the worst case a member must have $\log_2(n/2)$ keys for that resource. Also it is possible to monitor the Resource Lists (*RL*s) and if some *RL*s have exactly the same members, we can have the same *REK* for all such resources.

Let us consider the number of rekeying messages in our scheme. As the central tree implements efficient algorithms for *SDC*, the number of rekeying messages are low. Moreover, we create a tree for distributing *REK*s when needed in order to minimize the number of messages. As a result, the number of messages flying across the system is minimal.

The scheme is certainly scalable, since the central key tree for *SDC* can be used for any number of users. Also there can be any number of resources in the system as the *GC* just maintains a list of users subscribing to each resource and then uses the efficient recursive algorithm discussed in Section 4. The number of trees is just m + 1 and not 2^m - 1 + m as in other schemes[17]. Scalability of our scheme stems from the fact that the number of trees is linearly proportional to the number of resources, whereas other schemes[13] have exponential number of trees.

Compactness of our scheme due to linear relations makes it practical. Also our scheme is simple and can be completely automated. There are no complexities and ambiguities involved as in other Schemes.

CONCLUSION

We proposed a new yet effective scheme for key management for Differential Access Control in Dynamic Environments, based on principles of *SGC* Key Management and *SDC*. We compared our scheme with existing schemes, as a matter of fact there are not many schemes for the domain of this paper. We discussed the efficiency and scalability of our scheme in terms of number of trees, keys and rekeying messages, comparing with other schemes. Our scheme is compact, efficient, practical, and generic. The scheme scales better in a dynamic scenario when users change subscriptions or service providers add or revoke resources. The average number of shared keys found experimentally is far less than the one found theoretically. The scheme can be easily extended for resources with hierarchy. The future work includes 1) Checking system performance with a real large number of users and resources; 2) Testing the effect of user joins and leaves in a bursty manner on the system; and 3) Using n-ary tree instead of the binary tree used for illustration. The goal is to figure out the best value for n so that the Key Tree is efficient.

ACKNOWLEDGEMENT

This work was partially supported by the U.S. NSF grant CCR-0311577.

REFERENCES

1. Rafaeli, S. and D. Hutchison, 2003. A survey of keymanagement for secure group communication. ACM Computing Surveys (CSUR). 35: 309-329.
2. Wong, C.K., M. Gouda and S.S. Lam, 1998. Secure group communications using key graphs. SIGCOMM '98, Also University of Texas at Austin, Computer Science Technical report TR 97-23, pp: 68-79.
3. Zou, X., B. Ramamurthy and S. Magliveras, 2002. Efficient key management for secure group communication with bursty behavior. Proc. Intl. Conf. on Communication, Internet and Information Technology (CIIT), pp: 148-153.
4. Dondeti, L.R., S. Mukherjee and A. Samal, 1999. A dual encryption protocol for scalable secure multicasting. In 4th IEEE Symp. on Computers and Communications, pp: 2-8.
5. Steiner, M., G. Tsudik, and M. Waidner, 1996. Diffie-hellman key distribution extended to group communication. ACM Conf. on Computer and Communications Security (ACM CCS 1996), New Delhi, India, pp: 31-37.
6. Zou, X. and B. Ramamurthy, 2004. A block-free tgdh key agreement protocol for secure group communications. Proc. Intl. Conf. on Parallel and Distributed Computing and Networks, Innsbruck, Austria, pp: 288-293.
7. Abdalla, M., Y. Shavitt and A. Wool, 2000. Key management for restricted multicast using broadcast encryption. IEEE/ACM Trans. on Networking (TON), pp: 443-454.
8. Attrapadung, N., K. Kobara and H. Imai, 2003. Broadcast encryption with short keys and transmissions. Proc. 2003 ACM Workshop on Digital Rights Management, pp: 55-66.
9. Tzeng, W.G., 2002. A time-bound cryptographic key assignment scheme for access control in a hierarchy. IEEE Trans. on Knowledge and Data Engineering, pp: 182-188.

10. Wool, A., 2000. Key management for encrypted broadcast. ACM Trans. on Information and System Security (TISSEC), pp: 107-134.
11. Moyer, M.J., J.R. Rao and P. Rohatgi, 1999. A survey of security issues in multicast communications. IEEE Network, pp: 12-23.
12. Sandhu, R.S. and P. Samarati, 1994. Access control: Principles and practice. IEEE Commun. Mag., pp: 40-48.
13. Zhang, X., J. Park, F. Parisi Presicce and R. Sandhu, 2004. A logical specification for usage control. Proc. 9th ACM Symp. on Access control Models and Technologies, pp: 1-10.
14. Birget, J.C., X. Zou, G. Noubir and B. Ramamurthy, 2001. Hierarchical access control in distributed environments. IEEE Intl. Conf. on Commun. (ICC), pp: 101-140.
15. Chang, C.C., C.-H. Lin, W. Lee and P.-C. Hwang, 2004. Secret sharing with access structures in a hierarchy. 18th Intl. Conf. on Advanced Information Networking and Applications (AINA'04), Vol. 2.
16. Zou, X., B. Ramamurthy and S. Magliveras, 2001. Chinese remainder theorem based hierarchical access control for secure group communications. Lecture Notes in Computer Science (LNCS), Springer-Verlag (Intl. Conf. on Information and Communication Security), pp: 381-385.
17. Sun, Y. and K.J.R. Liu, 2004. Scalable hierarchical access control in secure group communications. Proc. IEEEINFOCOM.
18. Eskicioglu, A.M., S. Dexter and E.J. Delp, 2003. Protection of multicast scalable video by secret sharing: Simulation results. Proc. SPIE Security and Watermarking of Multimedia Content V, Santa Clara, CA, USA.
19. Caronni, G., K. Waldvogel, D. Sun and B. Plattner. 1998. Efficient security for large and dynamic multicast groups. Proc. 7th IEEE Intl. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98) (Cat. No.98TB100253). Los Alamitos, CA, USA.
20. Noubir, G., 1998. Multicast security. European Space Agency, Project: Performance Optimization of Internet Protocol Via Satellite.
21. Zou, X., S. Magliveras and B. Ramamurthy, 2004. Key tree based scalable secure dynamic conferencing schemes. to appear in proceedings of International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), MIT Cambridge, MA, USA.
22. Zou, X., B. Ramamurthy and S. Magliveras, 2004. Secure Group Communication Over Data Network. Springer.