

Efficient Algorithms for Secure Multicast key Management

Elham Khabiri , Saïd Bettayeb
School of Science and Computer Engineering
University of Houston-Clear Lake
elhamrose@yahoo.com, bettayeb@uhcl.edu

Abstract

Group Multicast is becoming a prevalent issue in Network applications such as teleconferencing, pay-per-view, and information services. In order to secure group communications by providing confidentiality and trustworthiness of messages, various methods were proposed in the literature. In this paper, we compare some of these methods. In particular, we consider the exclusive basis systems, and the batch re-keying methods. We also simulate the process of sub-group eviction for Exclusion Basis System (EBS) and we compare the work that needs to be done in a group-join and a group-eviction both in EBS and in Batch re-keying process. The comparisons of EBS with algorithms for Batch Re-keying process show that for group eviction Batch process is more efficient while EBS is for group join.

Keywords: group-eviction, group-join, batch re-keying process, Exclusion Basis Systems (EBS)

1. Introduction

Group Multicast has been used in many applications like pay-per-view services and secure teleconferencing. The goal is to make multicasting as secure and efficient as possible. Membership fluctuations need to be dealt with in an efficient way. That is, eviction/join of the users from/to the group. In the eviction process, we use a technique to change the keys that are known by the evicted user and send the new keys to the ones who are using those keys and are still group members. In the join process, we add users to the multicast group to be able to send them messages that are supposed to be sent to the group members. Let n be the number of people evicted from the group. We refer to this sub-group as the evicted sub-group. We could consider a group as a set of individuals. That is, send a message to each member of the group one by one. Since the unicast communication is technically well understood, we can use it in a group communication which is a

brand-new area of study. Such an extension is not scalable to large groups. [3]

In the unicast method, we have a client and a server that use an authentication protocol to communicate with each other. A symmetric key is created and shared by them to be used for pair wise secure communications. This procedure is extended to a group as follows. The group information is given to a trusted group server which manages the group access control. When a client wants to join the group, the client and group server mutually authenticate using an authentication protocol. After the authentication and acceptance into the group, each member shares with the group server a key, to be called the member's individual key.

For group communications, the group server distributes to each member a group key to be shared by all members of the group. For a group of n members, distributing the group key securely to all members requires n messages encrypted with individual keys (a computation cost proportional to group size n). Each such message may be sent separately via unicast. Alternatively, the n messages may be sent as a combined message to all group members via multicast. Either way, there is a communication cost proportional to group size n (measured in terms of the number of messages or the size of the combined message).

Consider a group server that creates a new group key after every join and eviction. After a join, the new group key can be sent via unicast to the new member (encrypted with its individual key) and via multicast to existing group members (encrypted with the previous group key). Thus, changing the group key securely after a join is not too much work.

After an eviction, however, the previous group key can no longer be used and the new group key must be encrypted for each remaining group member using its individual key. Thus, we see that changing the group key securely after an eviction incurs computation and communication costs proportional to n , the same as initial group key distribution. That is, large groups whose members join and leave frequently pose a scalability problem. [3]

In section 2, we introduce the previous work done for group multicast.

2. Previous work

2.1. Tree Structure

To implement secure multicast transmissions using encryption, each group member needs to know the session key which is the main key of the whole group. This session key is used by the multicast host to encrypt data packets before transmitting it to the whole group. For the privacy of the group, when a member is evicted from the group, the session key known by this member should be changed to a new value, disabling the evicted user to decrypt the messages that are sent to the group members. This new value of the session key should be sent to the members in a way that all of them except the evicted one could decrypt it.

If we have a group with a simple format of one session key and several personal keys for each user, we should send $n-1$ messages for a group of n members to transmit the new value of session key to other members. We call this message a Re-Keying message. For example if user u_3 is evicted from the group (see figure 1), the session key of k_{123} should be changed to a new value, say k_{12} . In this method, each member keeps only two keys: a session key and a personal key.

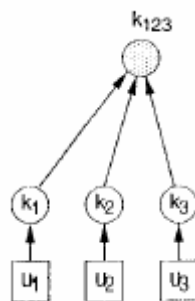


Figure 1 A group with 3 members and one session key [3]

In a hierarchical structure like a tree, we have some auxiliary keys, which divide a group into several subgroups. We call these auxiliary keys as Administrative keys. The administrative keys are used only for rekeying operations that take place when group membership changes [1]. This structure would work as follows:

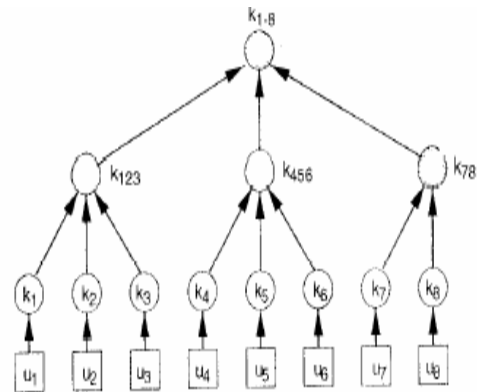


Figure 2 Tree structure for multicasting [3]

Each user keeps his/her personal key and all the auxiliary keys to the root. For example u_1 knows these keys: k_1, k_{123}, k_{1-8} . Therefore, when u_1 is evicted, these keys should be changed and the new values should be multicast to all other group members. For n users, with a tree of degree k we need only $(k-1)\log_k n$ re-keying messages and each user should keep $\log_k n$ keys, which equals to the height of the tree. The standard tree that is used here is binary tree.

2.2. Exclusion Basis System

A more general scenario of the binary tree structure discussed above is the Exclusion Basis System (EBS) in which the total number of keys that should be maintained by each user is smaller than the binary tree structure and the number of the rekeying messages can be half of the one in the binary tree. Morales et al. [1] define EBS as follows.

Definition. Let n, k and m be positive integers, such that $1 < k, m < n$. An *Exclusion Basis System of dimension (n, k, m)* , denoted by $EBS(n, k, m)$ is a collection Γ of subsets of $[1, n] = \{1, 2, \dots, n\}$ such that for every integer $t \in [1, n]$ the following two properties hold:

- (a) t is in at most k subsets in Γ , and
- (b) There are exactly m subsets, say A_1, A_2, \dots, A_m , in

Γ such that $\bigcup_{i=1}^m A_i$ is $[1, n] - \{t\}$. (That is, each element t is excluded by a union of exactly m subsets in Γ .)

To illustrate this, we describe $EBS(6, 2, 2)$. The collection of subsets $\Gamma = \{A_1 = \{1, 2, 4\}, A_2 = \{1, 3, 5\}, A_3 = \{2, 3, 6\}, A_4 = \{4, 5, 6\}\}$. In this simple example, we can easily verify that each integer t in the interval $[1, 6]$ is in exactly two subsets of Γ , and each integer t is excluded by a union of exactly 2 subsets in Γ . We

borrow the second example and its illustration from [1].

In EBS(8,3,2) the collection of subsets is

$$\Gamma = \{A1 = \{5,6,7,8\}, A2 = \{2,3,4,8\}, A3 = \{1,3,4,6,7\}, A4 = \{1,2,4,5,7\}, A5 = \{1,2,3,5,6,8\}\}.$$

One can easily verify that each integer $t \in [1,8]$ is in exactly 3 subsets in Γ , and each integer t is excluded by a union of exactly 2 subsets in Γ , as illustrated below:

$$\begin{aligned} [1,8] - \{1\} &= A1 \cup A2, \\ [1,8] - \{2\} &= A1 \cup A3, \\ [1,8] - \{3\} &= A1 \cup A4, \\ [1,8] - \{4\} &= A1 \cup A5, \\ [1,8] - \{5\} &= A2 \cup A3, \\ [1,8] - \{6\} &= A2 \cup A4, \\ [1,8] - \{7\} &= A2 \cup A5, \text{ and} \\ [1,8] - \{8\} &= A3 \cup A4. \end{aligned}$$

An Exclusion Basis System Γ of dimension (n,k,m) represents a situation in a secure group where there are n users numbered 1 through n , and where a key server holds a distinct key for each subset in Γ . [1]. Here the terms “key” and “subset” have the same concepts. We can map m to the number of rekeying messages and k to the number of keys that is maintained by each user.

If the subset A_i is in Γ , then the key A_i is known by each of the users whose number appears in the subset A_i . (For example, in the EBS(8,3,2) instance above, key A_1 is known by users 5, 6, 7, and 8 and by no others.)

Furthermore, for each $t \in [1,n]$ there are m sets in Γ whose union is $[1,n] - \{t\}$. From this it follows, as we shall see, that the key server can evict any user t , rekey, and let all remaining users know the replacement keys for the k keys they are entitled to know including the session key, by multicasting m messages encrypted by the keys corresponding to the m sets whose union is $[1,n] - \{t\}$.

To illustrate, consider the case when user 1 is ejected in the example EBS(8,3,2) above. User 1 knows keys A_3, A_4 , and A_5 , so these keys need to be changed and the new values sent out to authorized users. Observe that $[1,8] - \{1\} = A_1 \cup A_2$, so we show that two messages, encrypted by keys A_1 and A_2 , respectively, are sufficient to distribute the new keys to authorized users. Let the first message be one encrypted with key A_1 , and which contains four subparts.

The four parts of the message are:

- (1) a new session key, S' ,
- (2) replacement key for A_3 encrypted by the former A_3 key,
- (3) replacement key for A_4 encrypted by the former A_4 key,
- (4) replacement key for A_5 encrypted by the former A_5 key,

In other words, the first message is represented as:

$$A1(S', A3(A'3), A4(A'4), A5(A'5)),$$

where $A_i(x)$ denotes encryption of x by key A_i , and A'_i represents the replacement key for the old key A_i . In this notation, the second message would be:

$$A2(S', A3(A'3), A4(A'4), A5(A'5)),$$

It is easily verified that these two messages allow every remaining user, after user 1's departure, to learn exactly the set of new keys to which he is entitled. Furthermore, user 1 cannot decipher the rekey messages since user 1 does not possess keys A_1 and A_2 . At the conclusion of the rekey operation, user 1 has been effectively excluded from the secure group. The general case for an arbitrary EBS(n,k,m) is done in an analogous fashion. That is, for A_1, A_2, \dots, A_m in Γ

such that $\bigcup_{i=1}^m A_i$ is $[1,n] - \{t\}$, the key server can evict

user t by rekeying, and sending out m messages, such that the i^{th} message is encrypted with key A_i and contains the new session key and new administrative keys encrypted by their predecessors to limit their decipherability to appropriate users only. [1]

There is always a trade off between the number of rekeying messages needed to multicast and the number of the keys that should be remembered by each user. This is illustrated in Figure 3.

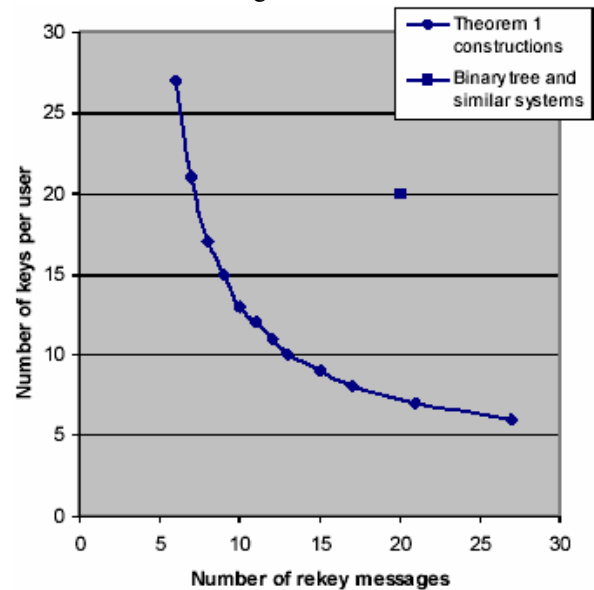


Figure 3 Trade off between k and m [1]

In [1], the authors showed that there is a positive solution to the EBS(n,k,m) problem if and only if

$$\binom{k+m}{k} \geq n.$$

2.3. Algorithms for Batch Re-keying process

The basic key management algorithms described briefly in the previous section are efficient for handling individual departures or arrivals, but not for cases where a large number of users either arrive or depart at the same time. In such cases, when k users arrive or depart simultaneously, the total number of multicast messages needed for re-keying would be k times the number of re-key messages for each individual. If k were, say, half of the total number of users, then $O(n \log n)$ messages would be sent. This is clearly inefficient, as a key management system could ignore the benefits offered by the tree and instead simply re-key through unicasting to each member. This would take $O(n)$ messages. [2]

Here we define how the batch processing works to give us better solutions than the previous methods for large number of departures and arrivals.

Consider G as set of administrative keys known by user g which is supposed to be evicted from the group. *Avoidance set* $A(G)$ for a subset G of users is defined as the set of nodes in a binary tree T which:

- (1) for every user g not in G , includes at least one node on the path from g to the root of T , and
- (2) for every user g in G , does not contain any node on the path from g to the root of T .

The basic idea is that, if G is a set of departing users, then the key manager needs to use administrative keys for encryption that are not known by any user in G and are such that every user not in G can decrypt one of the messages in order to learn the new replacement keys.

That is, the keys corresponding to the set $A(G)$ is a set of keys not known to the users in G , and these keys can be used to communicate to all remaining users.

We are interested in minimum size avoidance sets $A(G)$ for a given subset G .

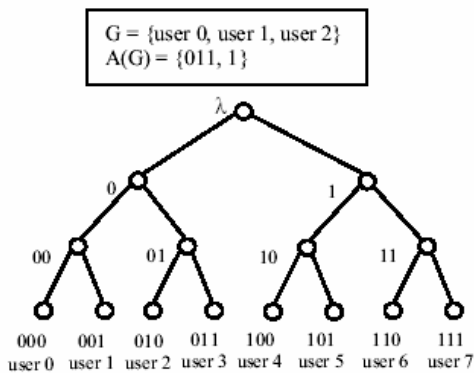


Figure 4

For the set of departing users $\{0,1,2\}$, a minimum size avoidance set $A(G)$ is $\{011, 1\}$ [2]

For example, in Figure 5 a binary tree T is shown with eight users. If $G = \{\text{user 0, user 1, user 2}\}$, then the minimum size avoidance set $A(G)$ consists of the nodes 011 and 1. That is, the keys corresponding to nodes 011 and 1 in T can be used to encode re-key messages which let all users, except those in G , know the new keys they need to know in order to continue. Furthermore, the two messages would contain all keys from 011 to the root (not including 011), and all keys from 1 to the root (not including 1). Specifically, the messages would be $E_{011}(\langle 01 \rangle \| \langle 0 \rangle \| \langle \lambda \rangle)$ and $E_1(\langle \lambda \rangle)$, where $\langle x \rangle$ denotes a new key to replace x and $E_y(z)$ denotes a message z encrypted by a key y , and λ represents the root of the tree.

A recursive algorithm called `AVOIDANCE_SET` is proposed in [2]. It computes the minimum $A(G)$ for any logical tree structure T and set of departing users G . The algorithm `AVOIDANCE_SET(T,x)` has two arguments: the first one, T , is the logical tree for the secure multicast group and the second one, x , is a node in the tree T . To compute $A(G)$ for the tree T and the set of users G , one calls the algorithm initially for $x = \Omega$, *i.e.* the root of the tree.

```

AVOIDANCE_SET(T,x)
if x is a leaf and is in G then stop
else Begin
if the subtree of T with root x has a
leaf that is in G
then Begin
AVOIDANCE_SET(T,left child(x));
AVOIDANCE_SET(T,right child(x))
End;
else Add x to the set A(G) and stop
End;

```

This algorithm works in time linear in the number of nodes in the tree and hence it is also linear in the number of leaves, which is the number of users in the secure multicast group. It also computes the minimum size avoidance set $A(G)$. [2]

In [2], the authors showed that the avoidance set $A(G)$ consists of at most $n/2$ nodes. They also proved that there is a set G of users for which $A(G)$ must contain $n/2$ users.

The Huffman code is used in [2] to create a new balanced, binary tree for key maintenance of n users in which a server sends a total of $2n-2$ messages, and where each message contains a single key.

The algorithm `NEW_TREE` is described below:

```

NEW_TREE( $\Omega$ )

```

```

begin
n ← | Ω |;
for i ← 1 to n-1 do
begin
allocate a new node z;
left(z) ← x ← Extract-MIN(Ω);
right(z) ← y ← Extract-MIN(Ω);
height(z) ← 1+max(height(x), height(y))
Insert(Ω, z)
end;
end;

```

That is, let Ω be the collection of all sub trees of T whose roots are elements of $A(G)$. [2]

3. Simulating EBS

We represent the process of EBS systems as a matrix. [1] The columns of the matrix represent the users and the rows represent the keys used in the whole system. For example if user1 knows the K3 and K4 in the EBS System the elements of A_{12} and A_{13} should be "1" and the rest ($A_{11}, A_{14}, \dots, A_{1T}$) should be zero. T which is the sum of the number of rekeying messages (m) and the number of keys k known by this user. The rows of the matrix are the binary strings of length $\binom{k+m}{k}$ where each column contains exactly k "1".

For example, if we have an EBS System that the number of keys remembered by each user is $k = 2$ and the number of re-keying messages $m = 2$ we would have the following matrix:

$$\begin{matrix}
 & U1 & U2 & U3 & U4 & U5 & U6 \\
 \begin{matrix} k1 \\ k2 \\ k3 \\ k4 \end{matrix} & \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}
 \end{matrix}$$

The following code computes matrix A.

```

Public Function CreateKey(ByVal Total As Integer, ByVal NumOfOne As Integer) As ArrayList
Dim i As Integer = 0
Dim InitStr As String = ""
Dim myStr As String = ""
Dim KeysArray As ArrayList = New ArrayList

For i=0 To Total-1
InitStr = String.Concat("0", InitStr)
Next

myStr = InitStr
If KeysArray.Count < Math.Pow(2, Total) - 2
Then
While NumberOfOne(myStr) < myStr.Length
myStr = increase(myStr)

```

```

If NumberOfOne(myStr) = NumOfOne Then
KeysArray.Add(myStr)
End If
End While
End If
Return KeysArray
End Function

```

This code starts from the initial string as an example: "0000" and counts to "1111". It increases each string value one by one and counts the number of ones for each string. If the number of ones in the string is k , then it adds it to the desired Array List. We use this ArrayList to create the matrix.

Finally, we have an array of strings that represent different permutations of ones in $k+m$ positions. By this method the array would be produced faster and takes less memory than using standard permutation generation algorithms. The order would be $O(n)$ in this method while the order would be $O(n^2)$ since it should use the factorial in each computation. When a user is evicted from the group, all other members who share some common keys with the evicted user should be informed about the new value of the changed keys. This could be done by the following algorithm:

```

For Each column in A Matrix belongs to Evicted User
j = index of row which the value of the matrix is one
For i=0 to Permutation (m+k, k)-1(The number of all users)
If Aij ="1"
Add the user to the ToBeInformedList
Next

```

In the above pseudo code the keys that are known by the evicted user should be changed. So, users that knew the keys should be informed of the value of the new keys. We can do that by a horizontal scan of the matrix. In each row the entries set to 1 represent the users that know the same keys. So we add them to a list called "ToBeInformedList".

As an example, suppose user1 and user2 of the matrix above should be evicted from the group.

Keys k3 and k4 that are known by user1 and hence, should be changed. A quick glimpse at rows k3 and k4 shows which users were using k3 and k4. In row k3, user3 and user5 are using this key (since they have 1 value) and in row k4, user2 and user4 are using this key. Therefore, if user1 and user2 are evicted from the group, user3, user5, and user4 should be informed of the new value of the changed keys.

4. Comparing the two methods

In group evictions, using EBS systems are not efficient and we would better use the algorithms for

batch re-keying. As an example, consider a tree in batch re-keying algorithm, with the height of 7 that can give service to 128 users. Each user would know 7 keys so we have $k = 7$. suppose that a group of 20 users should be evicted from the group. The number of re-keying message would be $2 \cdot A(G) - 2$ as stated before. $A(G)$ would be at most $n/2$ as proved in [2]. $2 \cdot n/2 - 2 = 126$ would be the number of reeking messages. In $EBS(10,7,3)$ which can serve $\binom{10}{7} = 120$

users, each user would send 3 re-keying messages. The number of re-keying message would be $m \cdot n' \cdot k$, where n' is the number of evicted users ($3 \cdot 20 \cdot 7 = 420$). On the other hand for group addition, if we want to add users to EBS system, it can be implemented by the following algorithm.

```
//Make different permutations of k
"1" in m+k positions.

If the number_of_users >  $\binom{m+k}{k}$ 

then ExtendMatrix

ExtendMatrix:
//Adds 0 to all other keys created
before
T = m+k
while m <> 1
    CreateKey(T,m-1) //makes new
    series of the keys
    m = m-1
end while
```

So, it would be much more efficient and simpler than using algorithms for batch re-keying process. Since after each extend part we need to add to the height of our tree since the parent node of each user should be changed.

5. Conclusion

We compared two different methods for group multicasting. The simplest solution to group multicasting is to consider the group as a set of individuals. The problem with this approach is that it is not scalable. The total number of administrative keys in the best EBS (n,k,m) system is the logarithm of the number needed for binary tree-based systems. Hence, we conclude that EBS system is much more efficient than binary trees. Furthermore, the number of needed re-keying messages in $EBS(n,k,m)$ system, when a user is evicted is also, in general, smaller than required in a binary tree-based system.

The comparisons of EBS with algorithms for Batch Re-keying process shows that for group eviction Batch process is more efficient while for group join EBS is. We also proposed two algorithms. A simple and more efficient way of constructing the matrix and one for creating a matrix in EBS while extended join occurs.

6. References

- [1] L. Morales, I. H. Sudborough, M. Eltoweissy, and M. H. Heydari, "Combinatorial Optimization of Multicast Key Management," Proceedings of the 36th Hawaii International Conference on System Sciences, 2003.
- [2] M. H. Heydari, L. Morales, and I. H. Sudborough, "Efficient Algorithms for Batch Re-keying Operations in Secure Multicast," Proceedings of the 39th Hawaii International Conference on System Sciences, 2006.
- [3] Chung Kei Wong, Mohamed Gouda and Simon S.Lam, "Secure Group Communications Using Key Graphs" presented at ACM SICCOMM, Vancouver, B.C., 1998.