A Scalable Distributed Subgroup Hierarchy Protocol For Secure Multicast Communication

Imane Aly Saroit[1], Said Fathy El-Zoghdy[2], and Mostafa Matar[3]

[1] Prof. Information Technology Dep., Faculty of Computers and Information, Cairo University, Egypt
[2(3)] Asist. Prof. (Lecturer). Computer Sciences Dep., Faculty of Science, Minufiya University, Egypt.
Email: iasi63@hotmail.com , Elzoghdy@yahoo.com, mattermostafa@yahoo.com

Abstract

In this paper, we propose an efficient and a scalable protocol for secure multicast communication. This protocol is based on the Iolus and the logical key hierarchy protocols. It divides the whole group into several subgroups as in the Iolus protocol. Each subgroup in turn is organized in a logical key hierarchy as in the LKH protocol. This decomposition reduces the complexity for a member join or leave form O(n) to O(log m), where n is the number of the whole group members and m is the number of each subgroup members. The performance of the proposed protocol is compared with that of the Simple App., Iolus and LKH protocols. The comparison is undertaken according to the computational overhead, communication overhead, storage overhead, and message size. The results show that the proposed protocol enhances the group performance in terms of computation overhead, and communication overhead especially at the leave operation.

Keywords: *Group Key Management, Secure Multicast, Logical Key Hierarchy, Decentralized Protocols, Distributed Protocols.*

1. Introduction

Multicasting is defined as delivering data from one sender to multiple recipients [18, 19, 20]. There are many multicast applications such as video conferences, Pay-Per-View TV, Internet stock quotes, software updates, etc. As a result of increasing multicast applications, data confidentiality becomes a big problem in secured multicast [18, 19, 21, 22]. In order to solve this problem a symmetric secret key, group key must be shared by the multicast members. These members have the authority to access the multicast data by using that group key.

Transferring data within the multicast occurs by using the secret group key twice; one for encrypting the data by the sender and the other for decrypting the ciphered data by the authorized receivers. With any change in the multicast resulted from member join or member leave a re-key operation for the group key must be occurred and then distributed to all multicast members [1, 3, 4, 5, 18]. As a result, the joining member can't access the multicast data sent before his joining the so-called *backward secrecy* and the leaving member can't access the multicast data sent after his leaving the so called *forward secrecy*.

Many protocols were proposed to solve the problem of group key distribution. These protocols were classified into four categories: Simple App., Centralized, Decentralized and Distributed key management approaches [1, 3, 4, 5, 8, 18, 23]

This paper proposes an efficient and a scalable protocol for secure multicast communication. This protocol is based on the idea of dividing the whole group into several subgroups as in the Iolus protocol. This decomposition reduces the complexity for a member join or leave from *O(n)* to *O(m)*, where *n* is the number of the whole group members and *m* is the number of each subgroup members. Each subgroup in turn is organized in a logical key hierarchy as in the LKH protocol which reduces the complexity for a member join or leave form *O(m)* to *O(log m)*. The group key is only known by the subgroup controllers that are represented by the root of each subgroup. The members

of each subgroup must know the appropriate subgroup key to make their decryptions. The subgroup key is organized using member secrets assigned to each member and server secrets assigned to each subgroup, and the inverse value of the member secrets is also used to manage the subgroup key at a member leave.

Each member in a single subgroup needs to store all the inverse values of the other members in that subgroup with the exception of its own. When a member joins a subgroup, the subgroup key is changed and sent to all existing subgroup members, where there is a join, via multicast. When a member leaves, the key server just sends its identity to the remaining members in its subgroup, and then they use the inverse value of the leaving member to update the subgroup key. The performance of the proposed protocol is compared with the performance of various previous protocols. The comparison is undertaken according to the *computational overhead*, *communication overhead*, *storage overhead,* and *message size*. The results show that the proposed protocol enhances the group performance in terms of computation overhead, and communication overhead especially at the leave operation which represents a big problem for most previous protocols.

This paper is organized as follows: Section 2 gives a brief summary for some of the previous related work. Section 3 presents a detailed description of the proposed protocol. Section 4 describes the results of numerical examination for the comparison between the performance of the proposed protocol and that of the simple App., LKH based protocols and Iolus protocol. Finally, Section 5 summarizes this paper.

2. Related work

As a result of the increasing of the multicast applications, the need to establish a group key becomes a vital requirement. Several solutions were proposed to solve the problem of the group key distribution. These approaches can be classified into four categories; Simple, Centralized, Decentralized and Distributed approaches [1, 3, 4, 5, 18, 19, 20, 21, 22].

In the simple approach, all of the group members share a secret group key assigned by the Key Distribution Center (KDC). For re-keying a group, the KDC encrypts the new group key by each user's individual key one by one, then constructs such a rekeying message including all these encrypted items. The re-keying operation is performed by just multicasting the re-keying message and each user only requires one time decryption upon receiving it [24].

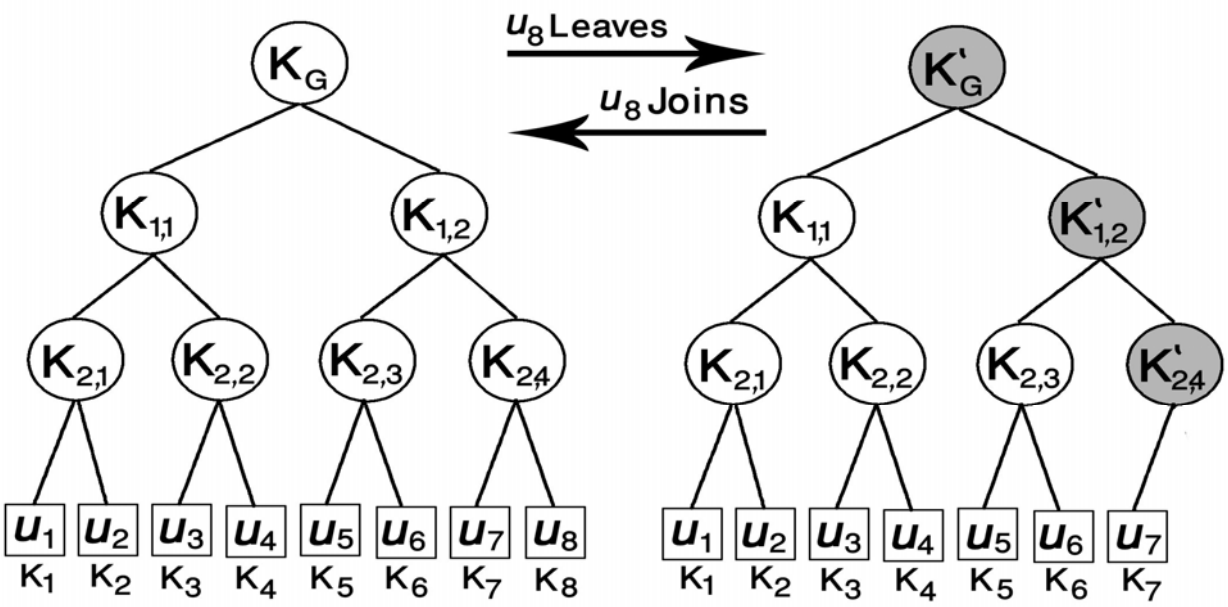
If a member joins a group on n members, in order to maintain the *backward secrecy*, the key server must renew the group key and distribute it to the group members. First, it encrypts the new group key by the old group key and multicasts it to the group members. Since the old group members know the old group key, they can decrypt the message and obtain the new group key. The new member u_{n+1} doesn't know the old group key, it cannot decrypt the key server's message. Instead, the key server sends the new group key to u_{n+1} via unicast encrypted by u_{n+1} individual key.

For a member to leave a group; suppose having a group of $n + 1$ members. If the member u_{n+1} leaves the group, the key server must renew the group key and distribute it to the remaining group members in order to maintain the *forward secrecy*. The key server cannot use the old group key to decrypt the new group key because u_{n+1} knows the old group key and can decrypt the key server's message by using it. Instead, the key server has to encrypt the old group key by each member's secret key. It generates a message and then transmits it to the whole group via multicast. On receiving the message, each member can recover the group key from the appropriate segment of the message using its own secret key [24].

In centralized approaches [5, 6, 7, 8] there is an entity plays the role of the group manager which is responsible for generating and distributing the group key to all members in the group. The centralized approaches are generally based on the idea of Logical Key Hierarchy (LKH). In LKH, a key distribution center maintains a key tree as shown in figure 1. The root of the tree plays the role of the group controller (GC) and shares a symmetric key with all members in the group, the internal nodes of the tree represent intermediate keys and the leaves represent the group members. Each member knows all keys from its leaf to the root; these keys will be called the *path keys* in this paper. For example, u_1 knows the set of keys $\{K_{2,1}, K_{1,1}, K_G\}$. It has to be noted that K_G is the group

key. If u_8 joins the group, as shown in figure 1 (a), the set of keys $\{K_{2,4}, K_{1,2}, K_G\}$ must be change into $\{K'_{2,4},K'_{1,2},K'_G\}$ respectively in order to maintain *backward secrecy*. The key server makes these changes and sends two messages; one to the existing members by multicast and the other to u_8 by unicast. The message sent by multicast is: $\{K'_{2,4}\}K_{2,4}$, $\{K'_{1,2}\}K_{1,2}$, $\{K'_G\}K_G$. u_7 can obtain $K'_{2,4}$ by decrypting the first part of the message using $K_{2,4}$, u_5,u_6 and u_7 can obtain $K'_{1,2}$ by decrypting the second part using $K_{1,2}$ and members from u_1 to u_7 can obtain K'_G by decrypting the third part using K_G, and the message sent by unicast is: $\{K'_{2,4}, K'_{1,2}, K'_G\}K_8$. u_8 can easily obtain the new keys by decrypting the message using K_8. If u_8 leaves the group as shown in figure 1 (b), the key server has to change $\{K'_{2,4}, K'_{1,2}, K'_G\}$ to maintain *forward secrecy*.

Since the leaving member knows the old group key and the old intermediate keys, the key server has to make a lot of encryptions. u_7 obtains the new keys by unicast, $\{u_5,u_6\}$ and $\{u_1,u_2,u_3,u_4\}$ obtain the new keys by decrypting the key server message using $K_{2,3}$ and $K_{1,1}$ respectively. The use of key tree will reduce the complexity from *O(n)* to *O(log n)*, where *n* represents the number of the whole group members. It has to be noted that centralized approaches suffer from the single point of failure problem, in addition for a large tree; the server's throughput can represent a bottleneck.



(a) An example of a member joins (b) An example of a member leaves

Figure 1: An example of hierarchical key tree

In the decentralized subgroup approach [9, 10, 11, 18, 19, 20, 21, 22], the whole group is split into small subgroups. Different controllers are used to manage each subgroup which minimizes the problem of concentrating the work on a single point. Membership changes treated locally which means that re-key of a subgroup doesn't affect the whole group which solves the scalability problem. The failure of one subgroup controller (SC) will not lead to the failure of the whole group. Iolus is an example of this approach. Mittra proposes Iolus [9], a framework with a hierarchy of agents that split the whole group into smaller subgroups. A Group Security Agent (GSA) manages each subgroup. GSAs themselves grouped in a top-level group that is managed by a Group Security Controller. Iolus uses independent keys for each subgroup and the changes that affect a subgroup are not reflected in other subgroups. Although Iolus is scalable, the SC (namely GSA) may become a bottleneck because the SC must decrypt the group messages and then re-encrypt it using the subgroup key.

In distributed approaches, the group key is generated in a contributory fashion, where all members contribute their own share in computing the group key. Examples of this approach can be found in [12, 13, 14]. The distributed approach is generally based on the Diffie-Hellman key exchange. The Diffie-Hellman exchange is used to establish a symmetric key between two entities.

Its security relies on the difficulty of finding logarithms for a large modulo prime number. Although this approach doesn't depend on a single entity to establish the group key, it suffers from the scalability problem. There are other proposed protocols based on the idea of mixing the advantages of the previous approaches, examples of these protocols can be found in [15, 16].

3. Proposed protocol

The proposed protocol is based on the idea of dividing the whole group into several subgroups as in Iolus protocol. This decomposition reduces the complexity for a member join or leave from *O(n)* to *O(m)*, where *n* is the number of the whole group members and *m* is the number of each subgroup members. Each subgroup in turn is organized in a logical key hierarchy as in the LKH protocol which reduces the complexity for a member join or leave from *O(m)* to *O(log m)*. The members in each subgroup contribute with each other to generate the subgroup key. This process delegates the key update process at a leave operation from the key server side to the member side.

The proposed protocol works in a hierarchy of two levels of controllers; the first for the group controller (GC) and the second is the subgroup controller (SC). The GC shares a symmetric key with all SCs which are trusted entities. The role of the SCs is to translate the data coming to their subgroups. Each SC works as the server of its subgroup. Figure 2 illustrates the structure of the proposed protocol.

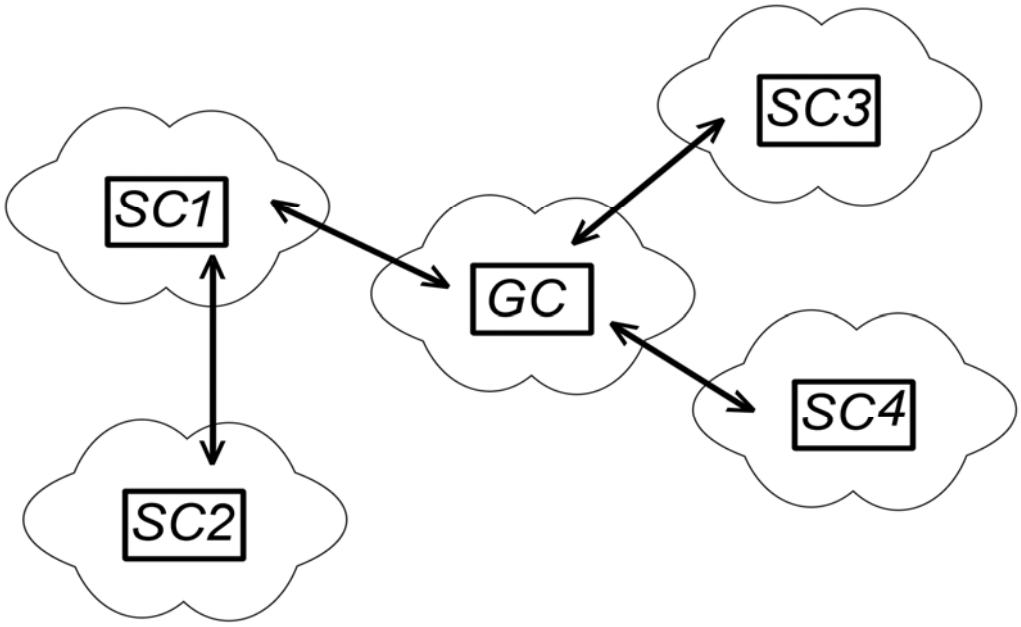


Figure 2: Structure of proposed protocol

The main objective is to establish a symmetric key between all group members in order to preserve the security of group communication. In case of a change occurs in the group membership by joining or leaving the group, the group key should be updated to maintain *backward secrecy* and *forward secrecy*

The structure of the subgroup hierarchy in the proposed protocol is shown in figure 4. The subgroup is organized in a hierarchy like the LKH approach and K_{SG1} is the key of the subgroup number *1*. For the operation of the proposed protocol, the following assumptions are made:

- SCs are trustable and static entities (i.e. they can join the group but they not allowed to leave the group).
- Members are organized in a hierarchy of a binary tree to enhance the key distribution at leave operation.

- The total number of subgroups is *n,* all subgroups are of height *h*, subsequently, each subgroup contains $m = 2^h$ users.
- Grouping operation is done taking into account the balance of the key distribution tree in each subgroup.
- The GC searches for an empty place in all subgroups when a new member wants to join the group. If the GC did not found an empty place, it would create a new subgroup where the new member will be allocated.

Figure 3 shows an example of a subgroup in the proposed protocol. From the hierarchy, we can see that the number of the subgroup members is $m=8$ and the height of the LKH is $h = log_2 8 = 3$.

1. u_1 and u_2 agree on the node key $K_{1,2}$.
   u_3 and u_4 agree on the node key $K_{3,4}$.
   u_5 and u_6 agree on the node key $K_{5,6}$.
   u_7 and u_8 agree on the node key $K_{7,8}$.
2. u_1, u_2 and u_3, u_4 agree on the node key $K_{1,4}$.
   u_5, u_6 and u_7, u_8 agree on the node key $K_{5,8}$.
3. u_1, u_2, u_3, u_4 and u_5, u_6, u_7, u_8 agree on the subgroup key K_{SG}.

The proposed protocol works as follows; the group key is only known by the SCs. The SCs share a symmetric key with the subgroup members this key is the subgroup key. The operation of the SCs is to decrypt the data come to their subgroups using the group key then they re-encrypt that data by their subgroup keys and send it encrypted to their associated subgroups members by multicast. The members of the group don't have to know the group key instead they have to know the key of the subgroup which they belong to in order to decrypt the ciphered data come to them from their associated SC. In the following paragraphs, we explain the operations of the proposed protocol in details.

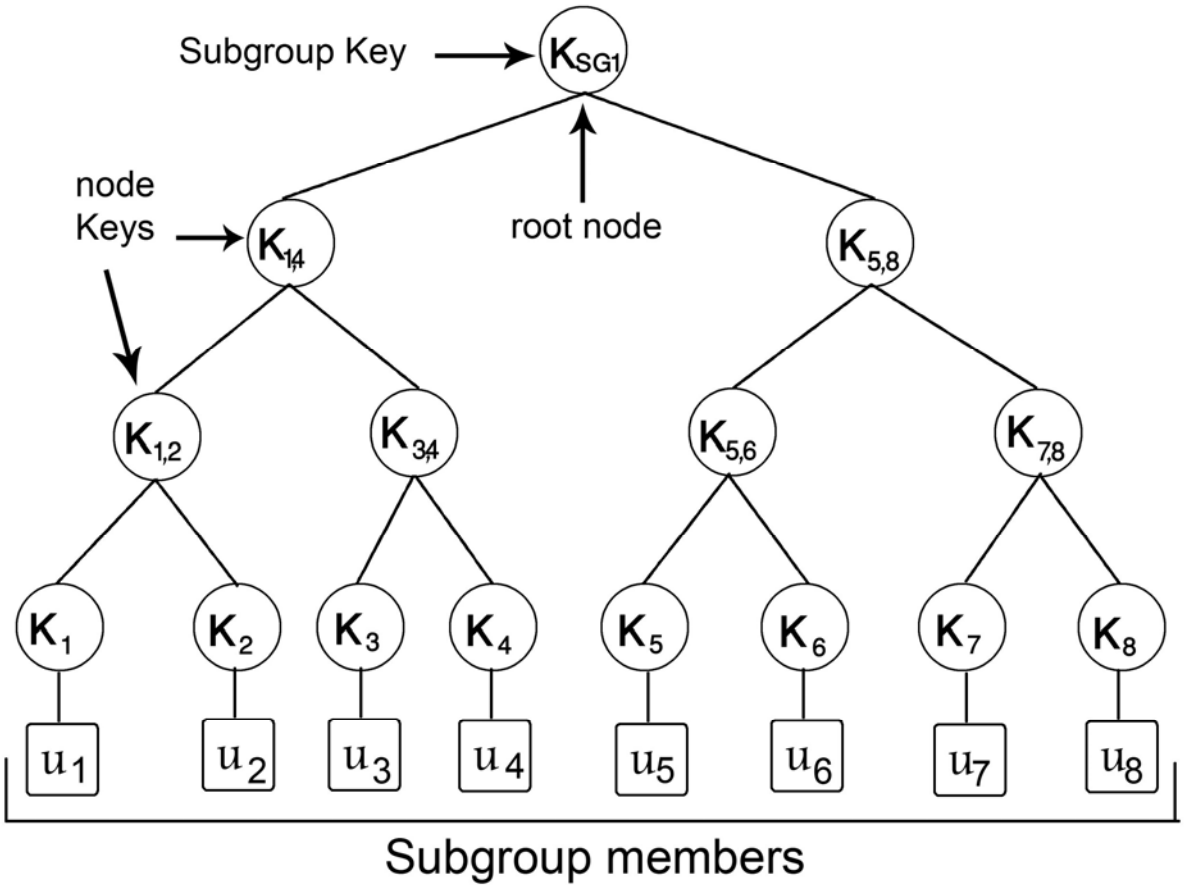


Figure 3: The structure of the subgroup hierarchy in the proposed protocol

3.1 Key Structure

The proposed protocol uses the modular exponential function as a one-way function. In this function, p is a large prime and g is a primitive element of multiplicative group Z^* [17], it is computationally different to determine α given g and g^{α} *(mod p)*. Based on this property, the subgroup key and the node keys are organized as follows:

1. The member secret a_j^i is selected under the condition that $2 \leq a_j^i \leq$p-2 and gcd $(a_j^s, p-1)=1$.
2. The server secret a_j^s is selected under the condition that $2 \leq a_j^s \leq$ p-2.

Using those secrets, the subgroup key for subgroup j is calculated by equation 1.

$$K_{SGj} \equiv g^{a_j^1 a_j^2 \ldots\ldots a_j^m a_j^S} \ (Mod\ p) \qquad (1)$$

The node keys are constructed by multiplying the exponents of its two child node keys in the logical key tree. This algorithm for rekeying can be illustrated using a simple example of a multicast subgroup of seven members. Figure 4 depicts the logical key tree for this subgroup.

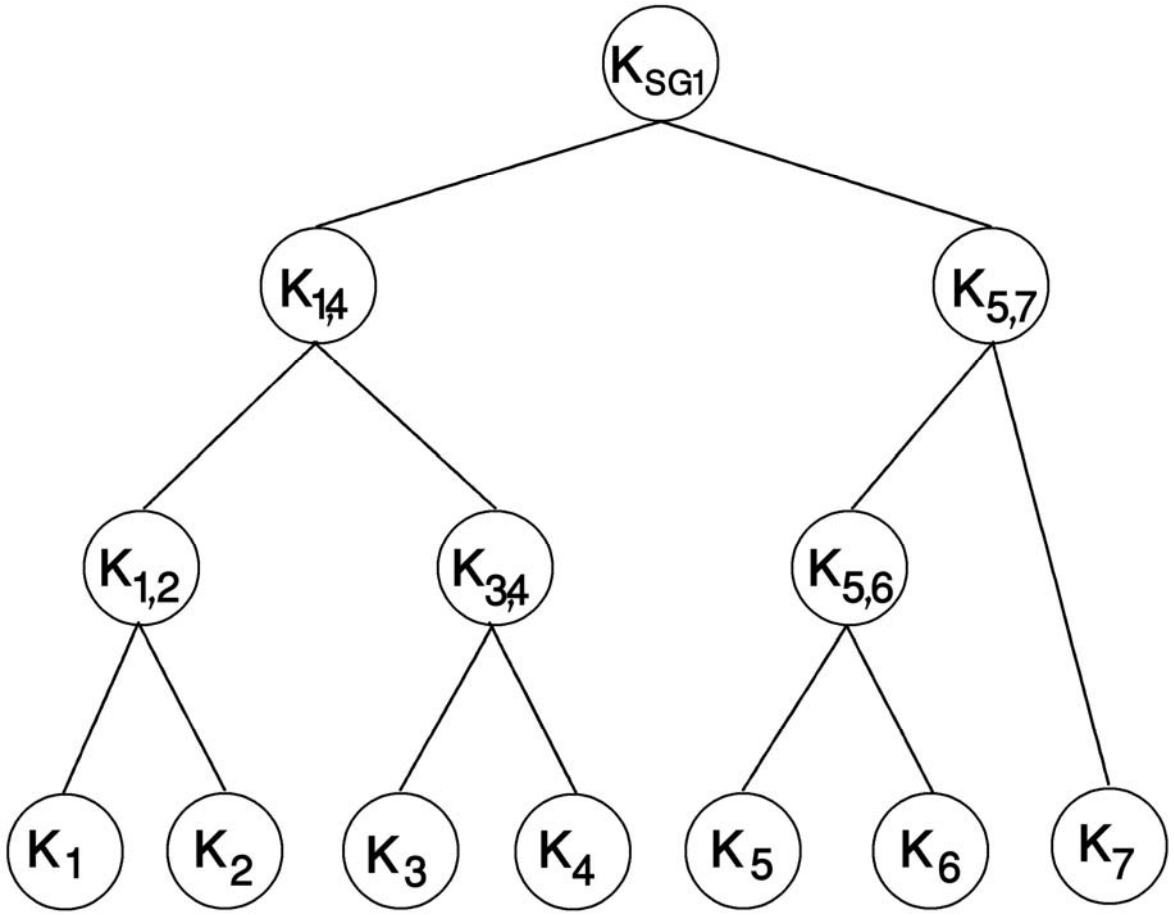


Figure 4: Subgroup example in the proposed protocol

Members u_1 and u_2 own keys K_1 and K_2 respectively, node keys $K_{1,2}$ and $K_{1,4}$, and the subgroup key K_{SG1}.

Members u_3 and u_4 own keys K_3 and K_4 respectively, node keys $K_{3,4}$ and $K_{1,4}$, and the subgroup key K_{SG1}.

Members u_5 and u_6 own keys K_5 and K_6 respectively, node keys $K_{5,6}$ and $K_{5,7}$, and the subgroup key K_{SG1}.

Member u_7 own keys $K7$, the node key $K_{5,7}$ and the subgroup key K_{SG1}.

In this situation, the keys are calculated as follows:

Level 1:

$$K_1 \equiv g^{a_1^1} \quad (Mod\ p)$$
$$K_2 \equiv g^{a_1^2} \quad (Mod\ p)$$
$$K_3 \equiv g^{a_1^3} \quad (Mod\ p)$$
$$K_4 \equiv g^{a_1^4} \quad (Mod\ p)$$
$$K_5 \equiv g^{a_1^5} \quad (Mod\ p)$$
$$K_6 \equiv g^{a_1^6} \quad (Mod\ p)$$
$$K_7 \equiv g^{a_1^7} \quad (Mod\ p)$$

Level 2:

$$K_{1,2} \equiv g^{a_1^1 a_1^2} \quad (Mod\ p)$$
$$K_{3,4} \equiv g^{a_1^3 a_1^4} \quad (Mod\ p)$$
$$K_{5,6} \equiv g^{a_1^5 a_1^6} \quad (Mod\ p)$$

Level 3:

$$K_{1,4} \equiv g^{a_1^1 a_1^2 a_1^3 a_1^4} \ (Mod\ p)$$
$$K_{5,6} \equiv g^{a_1^5 a_1^6 a_1^7} \ (Mod\ p)$$

When the SC receives the partial keys from its two child node keys he puts his secret value *a and* generates the subgroup key K_{SG1} using equation 2 and multicasts that key to all subgroup members.

$$K_{SG1} \equiv g^{a_1^1 a_1^2 \ldots\ldots a_1^7 a_1^S} \ (Mod\ p) \qquad (2)$$

3.2. Subgroup controller join

In the case of a SC *join*, suppose that a SC_{n+1} wants to *join* the group, the group key must be changed from $K_{(G)}$ into $K_{new(G)}$. The group controller sends the new group key to the existed SCs by broadcasting this message: $\{K_{new(G)}\}K_{(G)}$, $\{K_{new(G)}\}K_{(GC,SCn+1)}$, where $K_{new(G)}$ is the new group key, $K_{(G)}$ is the old group key and $K_{(GC,SCn+1)}$ is the symmetric key shared between the Group Controller GC and the new SC. The old SCs decrypt the first part of the message using the old group key which they already have to get the new group key. The joining SC decrypts the second part of the message using the symmetric key shared with him and the GC to get the new group key. Therefore, after joining of a new SC, only the subgroup controllers need to update the group key without affecting the whole group members. The GC makes two key encryptions; one for the old SCs and the other for the new SC. Therefore, the *join* of a new subgroup containing *n* members will require that the GC makes two encryptions and the whole group members not affected. Subsequently, better computation and communication performance can be achieved.

To illustrate the role of the SC in the data transmissions suppose that a sender wants to send a message to subgroup number *1*. The sender encrypts his message $\{M\}$ by using a key K_M and that key is also encrypted by using the group key K_G. The sender sends the following message to the SC1: $\{M\}K_M, \{K_M\}K_G$. When the SC1 received that message he uses the group key (K_G) to decrypt the second part of the message and get the K_M which he uses to decrypt the first part of the message to get the origin message and then encrypts it using the subgroup1 key (K_{SC1}) and multicast the ciphered message to his subgroup members.

3.3. Member join

If a member wants to join a group, he sends a '*join*' request to the GC who looks for a place to the new member and then directs him to a specified subgroup. The SC of the specified subgroup takes the *join* request from the group controller and makes changes to maintain the backward secrecy of his subgroup.

The following example illustrates this process. Suppose that member *u8* wants to *join* the group, he sends a '*join*' request to the GC who searches for an empty place and finds it in the subgroup number 1 (SG1) for example as shown in figure 5. The GC directs the new member to the SC1 who performs the following:

- Creates a new node key $K_{7,8}$, K_7 becomes its left child and the key of the new member K_8 becomes its right child.
- Assigns an identity to the new member.
- Assigns a secret value a_1^8 to the new member and calculate its inverse value a_1^{-8}.
- Changes his secret value a_1^s to change the subgroup key to maintain the backward secrecy.
- Updates the path keys of the new member.
- Sends the new keys and the inverse values of the existing members to the joined member via unicast.
- Sends the subgroup key and the inverse value of the new member to all subgroup members via multicast.

The SC1 sends the following message to the members in his subgroup: $\{K'_{SG1}, a_1^{-8}\}K_{SG1}, \{K_{5,8}\}K_{5,7}, \{K_{7,8}\}K_7, \{K'_{SG1}, K_{5,8}, K_{7,8}, a_1^{-1}a_1^{-2}.....a_1^{-7}\}K_8$. All existing members obtain the new subgroup key K'_{SG1} and the inverse value of the new member a_1^{-8} by decrypting the first part of the message using the old subgroup key K_{SG1}. The members u_5, u_6 and u_7 obtain the new node key $K_{5,8}$ by decrypting the second part of the message using the old node key $K_{5,7}$ and obtains the node key $K_{7,8}$ by decrypting the third part of the message,. The new member u_8 obtains his path keys and the inverse values of the other member's secrets by decrypting the last part of the message as shown in figure 5.

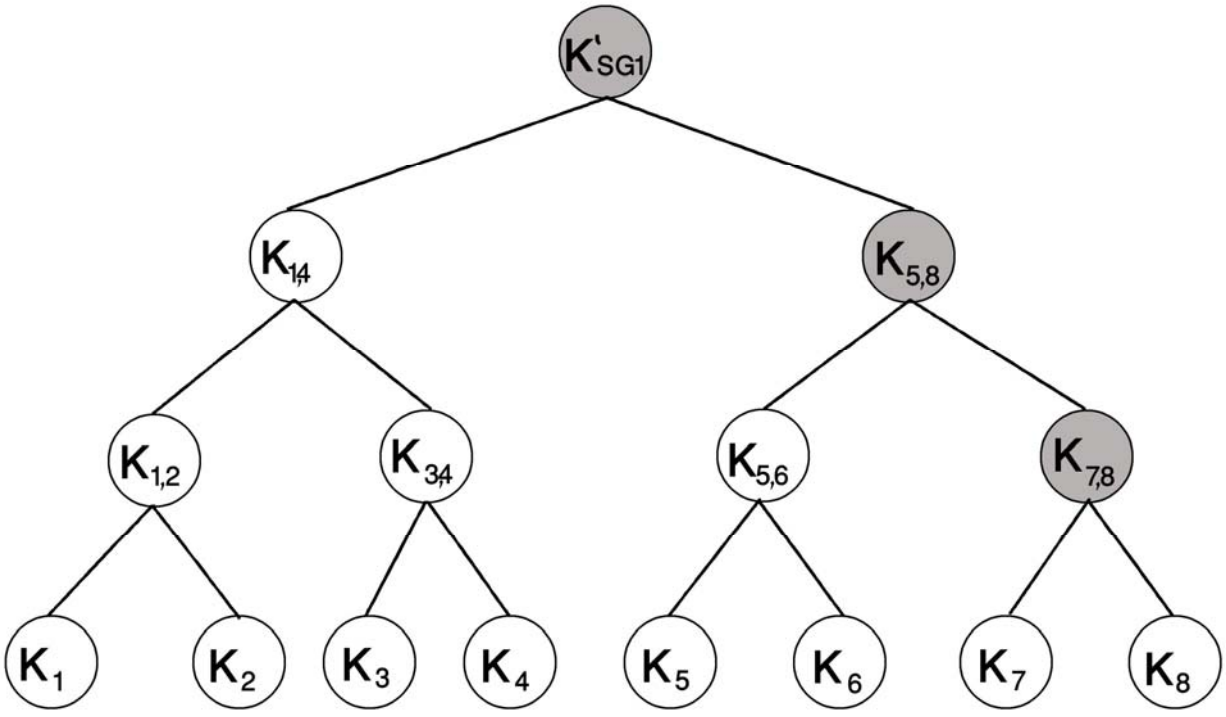


Figure 5: Keys structure of SG1 when u_8 joins it.

So the new keys will be updated as follows:

$$K_{7,8} \equiv g^{a_1^7 a_1^8} \quad (Mod\ p)$$

$$K_{5,8} \equiv g^{a_1^5 a_1^6 a_1^7 a_1^8} \quad (Mod\ p)$$

$$K'_{SG1} \equiv g^{a_1^1 a_1^2 \ldots\ldots a_1^7 a_1^8 a_1^{S'}} \quad (Mod\ p)$$

3.4. Member leaves

If a member wants to leave the subgroup he sends a '*leave*' request to the SC of his subgroup. The SC sends the identity of the leaving member and the keys which must be updated to the remaining subgroup members via multicast. When the members received the SC's message they use the inverse value of the leaving member to update the keys in the message.

In our example, if u_8 wanted to leave the SG1, the keys K_{SG1}, $K_{5,8}$, $K_{7,8}$ would be updated to maintain forward secrecy. According to our protocol, updated keys need not to be sent to the remaining members. Instead, the SC1 just prepare one message for subgroup 1 indicating u_8 leaves and the keys must be updated.

The SC1 sends the message $\{u_8, K'_{SG1}\}\ K'_{SG1}, \{K_{5,8}\}\ K_{5,8}$. When the remaining members receive this message and find that the keys in the message are encrypted by the same keys they know that there is a leave operation and the identity of the leaving member is 8.

They use the inverse value of u_8 (a_1^{-8}) to update the sent keys. So the new keys will be:

$$K''_{SG1} \equiv (K'_{SG1})^{a_1^{-8}} \quad (Modp)$$

$$K''_{SG1} \equiv g^{a_1^1 a_1^2 \ldots\ldots a_1^7 a_1^8 a_1^{-8} a_1^S} \quad (Modp)$$

$$K''_{SG1} \equiv g^{a_1^1 a_1^2 \ldots\ldots\ldots\ldots a_1^7 a_1^S} \quad (Modp)$$

$$K'_{5,8} \equiv (K_{5,8})^{a_1^{-8}} \quad (Modp)$$

$$K'_{5,8} \equiv g^{a_1^5 a_1^6 a_1^7 a_1^8 a_1^{-8}} \quad (Modp)$$

$$K'_{5,8} \equiv g^{a_1^5 a_1^6 a_1^7} \quad (Modp)$$

As it is noticed, the key server doesn't generate new node and subgroup keys after a leave. Instead, it just sends the identity of the leaving member and the keys that must be updated to the remaining members. The remaining members use the inverse value of the leaving member to update these keys. In this way, updating the keys after a leave is shifted to member's side which improves the efficiency of re-keying at leave.

4. Protocol Evaluation

In this section, we compare the performance of the proposed protocol with that of simple App., LKH based protocols and Iolus protocol for *join* and *leave* operations. For the comparison to be conducted, the following general assumptions are considered:

- The total number of subgroups in both Iolus and the proposed protocol is N=16 subgroups.
- The total number of each subgroup members is m members. Subsequently, the total number of the whole group is $n = N \times m$ members.
- The height of the proposed protocol's tree is h, where $h = log_2 m$, i.e. the total number of each subgroup members is equal to $m = 2^h$.

The comparison will be undertaken according to the following criteria:

- **Computational overhead:**
  - Key generation overhead: the number of keys that must be generated at join and leave by the key server and a member node.
  - Encryption/Decryption overhead: the number of encryptions at the key server and the number of decryptions for a member node.
- **Communication overhead:** the number of transmissions from the key server.
- **Message size:** the number of keys in one multicast message.
- **Storage overhead:** the number of keys stored at the key server, and by a member node.

4.1. Computational overhead

Computational overhead can be divided into key generation overhead and encryption/ decryption overhead.

4.1.1. Key generation overhead

Key generation overhead is the number of keys that must be generated at join and leave events by the key server and a member node.

In the simple approach, the key server generates *two* keys; an individual key to the joined member and a new group key at a member join operation. At a member leave operation, the key server generates only *a new group* key and distributes it to the group members encrypted with their individual keys [24].

In the LKH protocol, when a new member joins a group, the key server generates *a session key* to the new member and new $log_2 n$ keys to maintain *backward secrecy*. When a member leaves a group, the key server generates $log_2 n$ keys [2].

In the Iolus protocol, when a new member joins a subgroup, the subgroup controller, which plays the role of the key server for its subgroup, generates only *two* keys. At a member leave operation, the subgroup controller generates only *a new subgroup* key and distributes it to its subgroup members [9].

In the proposed protocol; when a new member joins a subgroup of m members. The subgroup controller gives *a session* key, a secret value and the alternative key of its subgroup to the new member. Then it changes all the intermediate node keys from the node of the joining member to the root key which represents the subgroup key by factoring and changing its secret value in the subgroup key and the intermediate node keys after adding the new member's secret value to them. Consequently, the subgroup controller generates $log_2 m$ keys, in addition with the *session* key and the secret value of the joining member.

At a member leave operation the subgroup controller *doesn't generate* any keys. Instead it multicasts the identity of the leaving member to all the subgroup members to be factored from the subgroup key by using the leaving member's inverse value.

Table 1 shows the time complexity of key generation overhead at the key server and member nodes along the path at join and leave operations for all the examined protocols. Figure 6 shows the number of keys generated at the key server at join operation.

From table 1 and figure 6, we can see that the simple approach and Iolus protocols have the smallest overhead of the number of key generated by the key server at a member join. Since the height of the subgroup tree h_{sgrp} in the proposed protocol is less than that of the LKH protocol $h_{LKH,}$ the proposed protocol has a less overhead than the LKH protocol. In the proposed protocol, the key server generates $log_2 m$ keys, at the same time it generates $log_2 n$ keys in the LKH protocol. Figure 7 shows the number of keys generated at the key server at *leave* operation. From that figure one can notice that the proposed protocol has the smallest overhead at the leave operation because the key server doesn't generate any keys in that case.

	Key server		Member nodes	
	Join	Leave	Join	Leave
Simple App.	0	0	0	0
LKH	$O(\log_2 n)$	$O(\log_2 n)$	0	0
Iolus	0	0	0	0
Our protocol	$O(\log_2 m)$	0	0	0

Table 1: Complexity of key generation overhead at the key server and a member node for join and leave operations

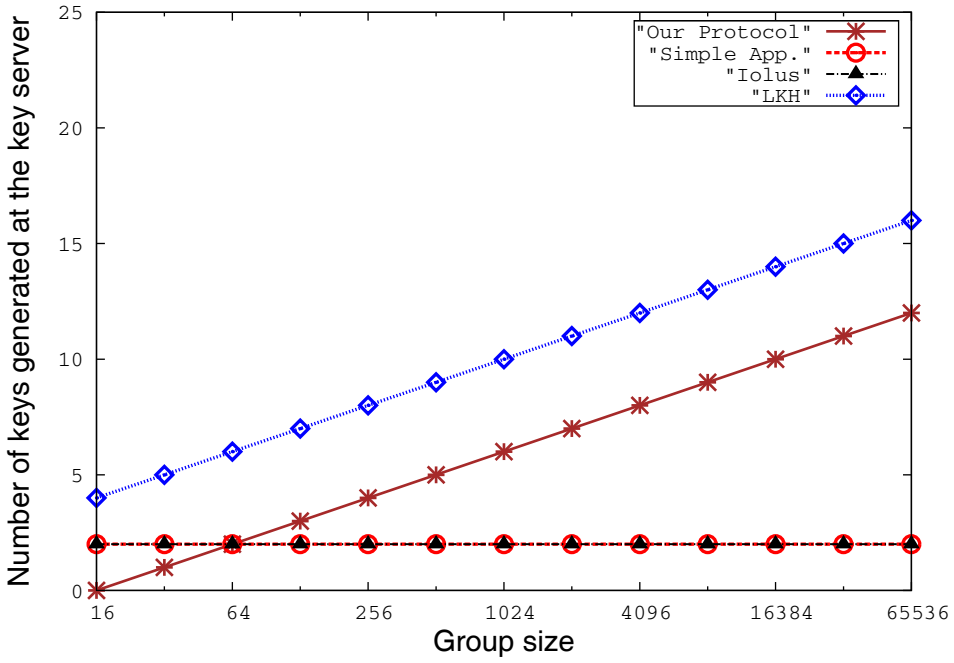


Figure 6: Number of keys generated at the key server *at join* operation

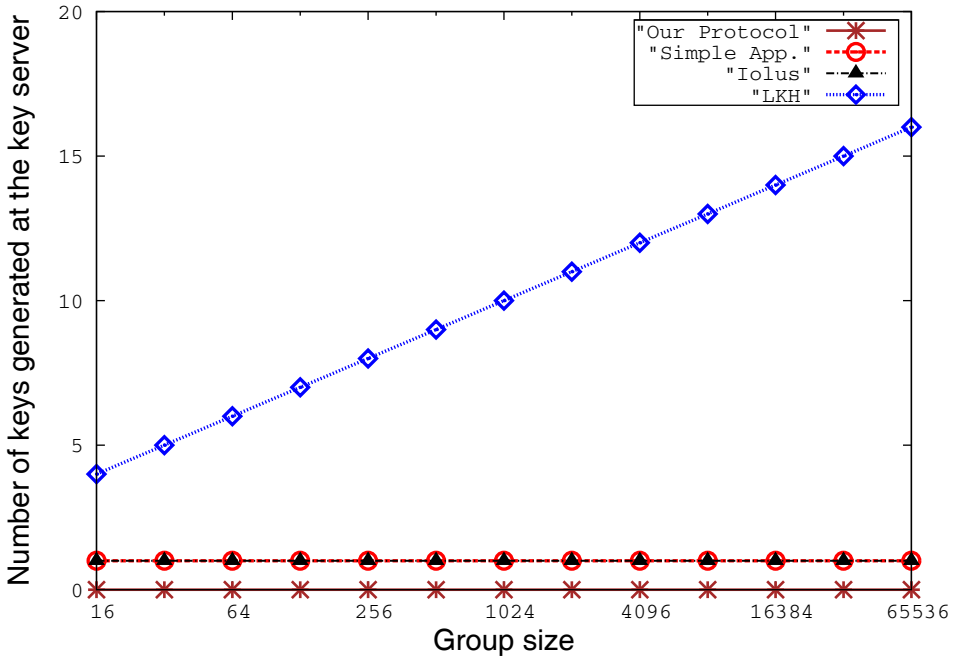


Figure 7: Number of keys generated at the key server *at leave* operation

4.1.2. Encryption/decryption overhead

Encryption/Decryption overhead is the number of encryptions at the key server and the number of decryptions for a member node.

In the simple approach, when a new member joins a group, the key server multicasts a new group key to all group members encrypted with the old group key. Further, the key server sends the new group key to the new member encrypted by its session key. So, the key sever performs *two encryptions* when a new member joins its group. The new member and the existing member make only *one decryption*. But, when a member leaves the group, the key server sends a new group key to the remaining *n-1* members encrypted with their individual keys. So, the key server makes *n–1 encryptions* at a member *leave* operation and each of the remaining members makes only *one decryption* [24].

In the LKH protocol, when a new member joins a group, the key server performs $3\log_2 n$ *encryptions.* At a member leave, the key server performs $2\log_2 n$ *encryptions*. Each member in the LKH protocol makes $\log_2 n$ *decryptions* at both a member *join* and member *leave* operations [2].

In the Iolus protocol, the key server makes *two encryptions* at a member join operation. At a member leave, the key server performs *m-1 encryptions*. Each member in this protocol performs only one decryption at both a member *join* and member *leave* operations [9].

The proposed protocol; is different from the LKH protocol in the number of encryptions per node. It uses a one-way function tree, the key server makes one encryption per node but in the LKH protocol it makes two encryptions per node. At a join operation, the key server must change the path keys of the joining member. The key server encrypts the new keys by its corresponding old keys. Since each subgroup of m members is organized in a hierarchical tree, the key server makes $\log_2 m$ *encryptions*. In order to obtain the new keys, the key server sends to the joining member its path keys encrypted by its individual key so that the key server makes other $\log_2 m$ encryptions for the joining member. So, the overall encryption performed by the key server at the join operation is equal to $2\log_2 m$.

When a member leaves a subgroup, the key server must change the path keys of the leaving member so it performs $\log_2 m$ encryptions. When the members of a subgroup receive the encrypted $\log_2 m$ keys from the key server, each member makes $\log_2 m$ decryptions in a member join operations, but it *doesn't make any decryptions* at a member leave operation as it only factors the leaving member's secret value from the necessary keys by using the inverse value of the leaving member.

Table 2 shows time complexity of encryption/decryption overhead at key server and number of decryptions at a member node at both *join* and *leave* operations. Figures 8(10) and 9(11) show the number of encryptions (decryptions) at the key server (a member node) at *join* and *leave* operations respectively.

At *join* operation the Simple App. and Iolus protocols have the smallest number of encryptions at the key server and number of decryptions at member nodes as shown from table 2 and figures 8, and 10. But the proposed protocol has the smallest number of encryptions and decryptions at the key server and member nodes respectively at the *leave* operation as shown from table 2 and figures 9 and 11.

	Key server		Member nodes	
	Join	Leave	Join	Leave
Simple App.	0	O(n)	0	0
LKH	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$
Iolus	0	O(m)	0	0
Our protocol	$O(\log_2 m)$	$O(\log_2 m)$	$O(\log_2 m)$	0

Table 2: Complexity of encryption/decryption overhead at the key server and a member node for join and leave operations

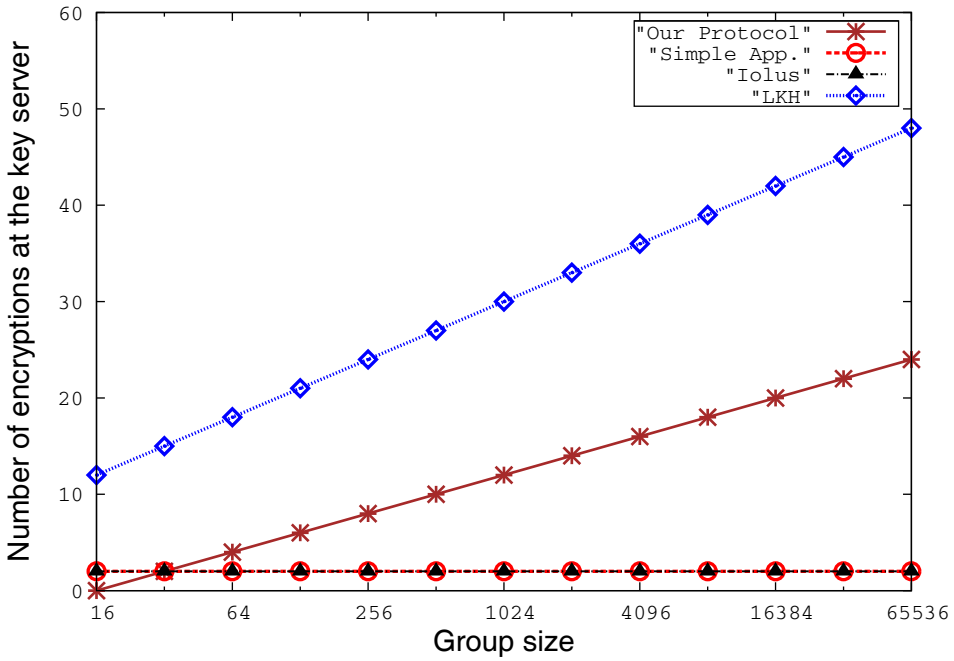


Figure 8: Number of encryptions at the key server *at join* operation

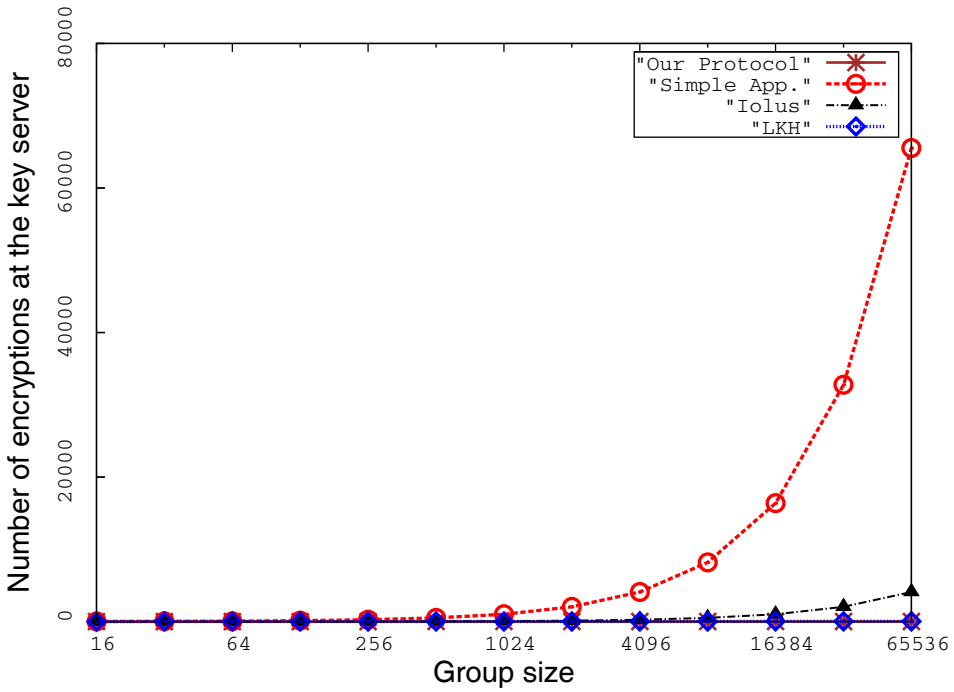


Figure 9: Number of encryptions at key server *at leave* operation

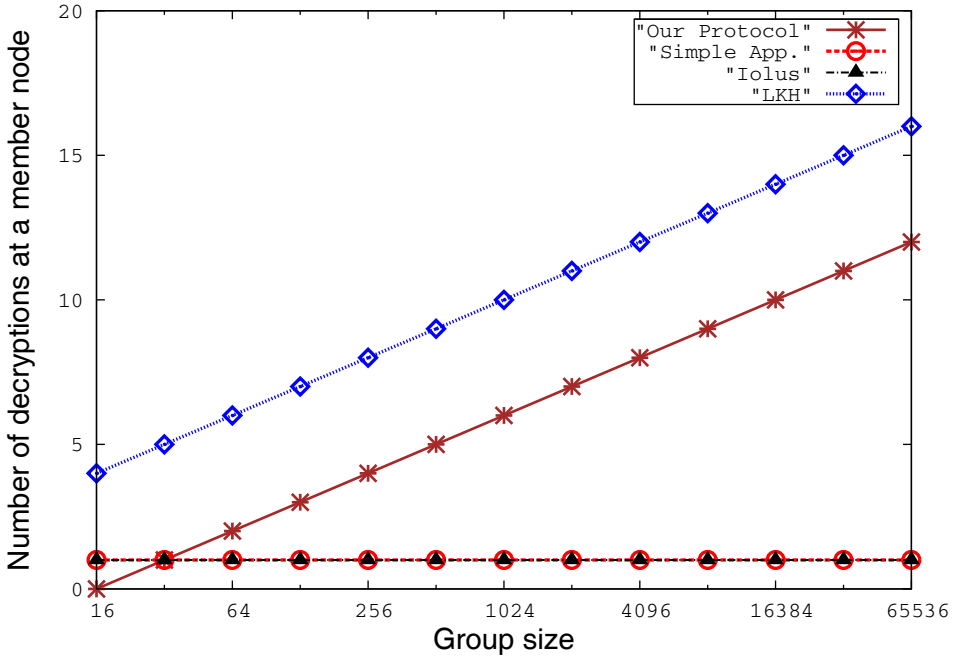


Figure 10: Number of decryptions at member nodes *at join* operation

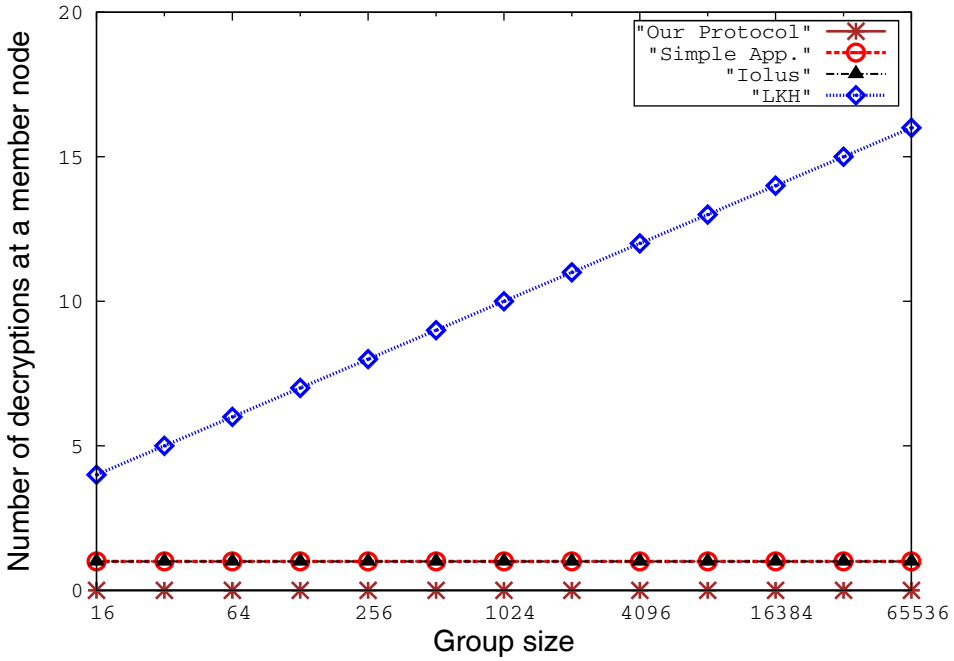


Figure 11: Number of decryptions at member nodes *at leave* operation

4.2. Communication overhead

The communication overhead is the number of transmitted control messages from the key server.

In the simple approach, when a new member joins the group, the key server sends *two control messages* including the new group key. One message to the existing group members and the other to the joining member. If a member leaves a group of *n* members, the key server encrypts the new group key with the individual keys of the remaining *n-1* group members and sends *n-1* control messages to them [24].

In the LKH protocol, when a member joins a group, the key server transmits $2log_2n-1$ messages to the group members. When a member leaves a group, the key server transmits at most $2log_2n$ messages to the remaining members [2].

In the Iolus protocol, at a member *join* operation, the subgroup controller multicasts *a message* to all subgroup members with the new subgroup key and sends *a message* with the new subgroup key to the joining member via unicast. At the *leave* operation, the subgroup controller transmits either *m-1* separate messages involved the new group key encrypted by each member individual key to the remaining members or transmits one message containing *m-1* copies of the new subgroup key each encrypted with a different individual key [9].

In the proposed protocol, when a new member joins a subgroup, the key server changes its path keys and transmits only *one message* involves the new keys encrypted by the old subgroup key to the existing subgroup members. When an existing member leaves a subgroup, its path keys must be changed. The key server transmits *one message* containing the identity of the leaving member. The remaining members use this identity to obtain the new keys.

Table 3 shows the time complexity of communication overhead for the key server at leave and joins operations. It is important to note that the control messages transmitted only from the key server, so considering member node in the table is meaningless. Figure 12 shows the number of transmitted control messages from the key server at leave operation.

	Join	Leave
Simple App.	0	O(n)
LKH	$O(\log_2 n)$	$O(\log_2 n)$
Iolus	0	O(m)
Our protocol	0	0

Table 3: Complexity of communication overhead at the key server for leave and join operations

From table 3 and figure 12, one can notice that the Simple App., Iolus and proposed protocol have the smallest communication overhead at *join* operation compared with the LKH protocol. At *leave* operation, the Simple App. has the biggest overhead and the proposed protocol has the smallest since the subgroup controller sends a single message at a member leave operation.

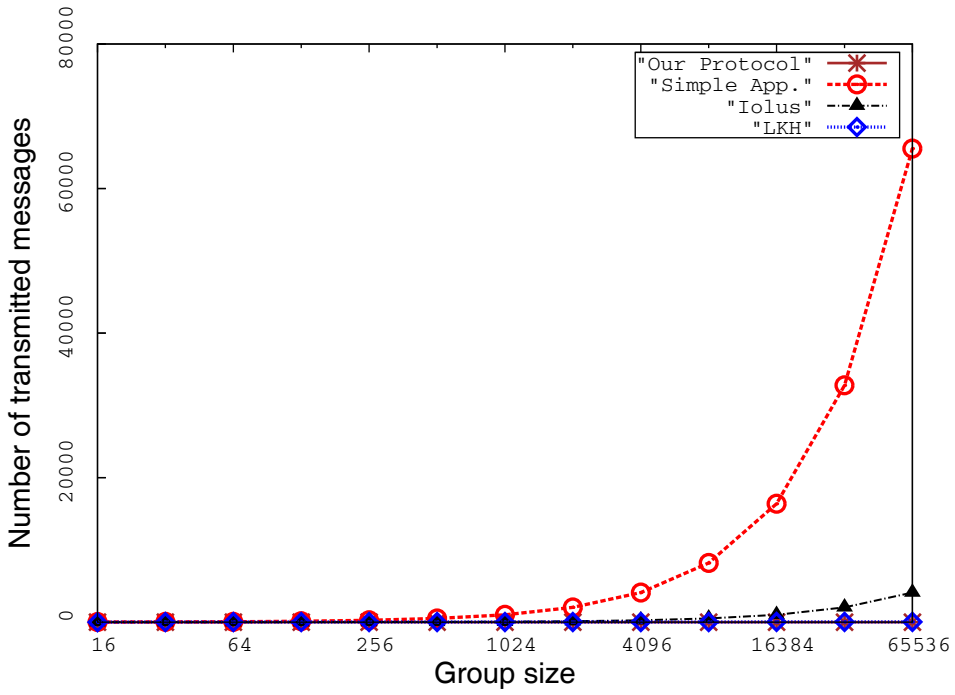


Figure 12: The communication overhead *at leave* operation

4.3. Message size

Message size is given by the number of keys included in one multicast message.

In the simple approach, only *two keys* are included in the message sent by the key server to all group members at a *join* operation; one encrypted by the old group key and the other encrypted by the new member's individual key. At *leave* operation, the key server multicasts a message of *n-1* keys that are the new group key encrypted by each member individual key [24].

In the LKH protocol, the number of the keys included in one multicast message at a member join operation is $2log_2n-1$ keys. The message size at a leave operation in the LKH protocol is $2log_2n$ keys [2].

In the Iolus protocol, the multicast message includes only *two keys* at a member join operation and it includes *m-1* keys at a member leave operations [9].

In the proposed protocol, the message contains log_2m+1 keys at a *join* operation; log_2m new path keys of the joining member and the inverse value of the joining member all encrypted by the old subgroup key. At a *leave* operation, the subgroup controller sends *a multicast* message including only the identity of the leaving member to all the subgroup members.

Table 4 shows time complexity of message size at the key server for join and leave operations in one multicast message. Figures 13 and 14 show the number of keys in one multicast message at the *join* and *leave* operations respectively.

	Join	Leave
Simple App.	0	O(n)
LKH	$O(log_2n)$	$O(log_2n)$
Iolus	0	O(m)
Our protocol	$O(log_2m)$	$O(Log_2m)$

Table 4: Complexity of message size at the key server for join and leave operations

As shown in table 4 and figures 13 and 14, the Simple App. and Iolus protocols have the smallest overhead at *join*. They have the biggest overhead at *leave* operation. The proposed protocol is better than the LKH protocol at *join* operation and it is the best protocol at *leave* operation.

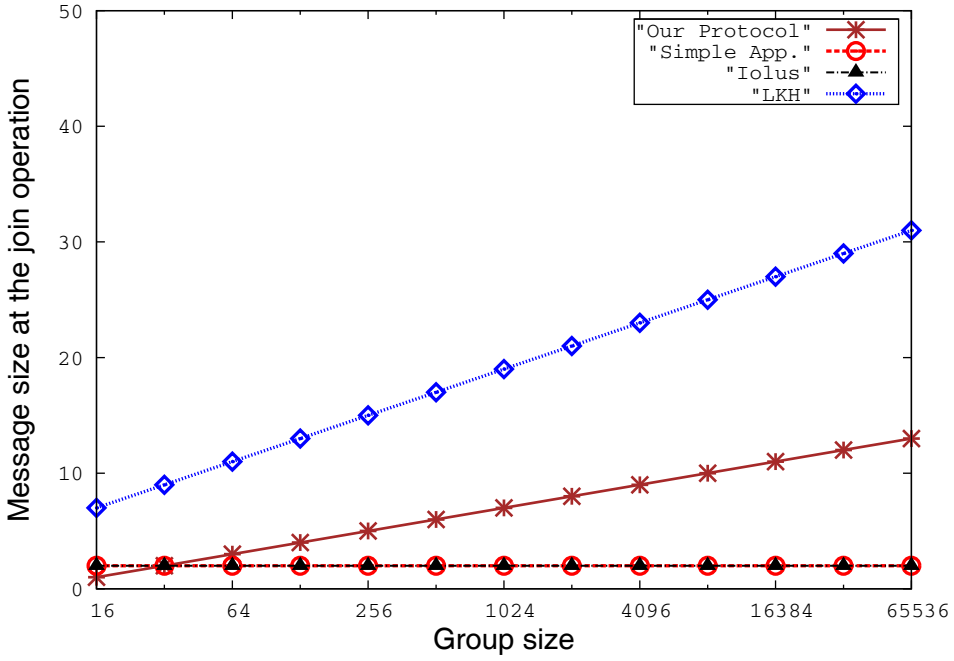


Figure 13: Message size overhead *at join* operation

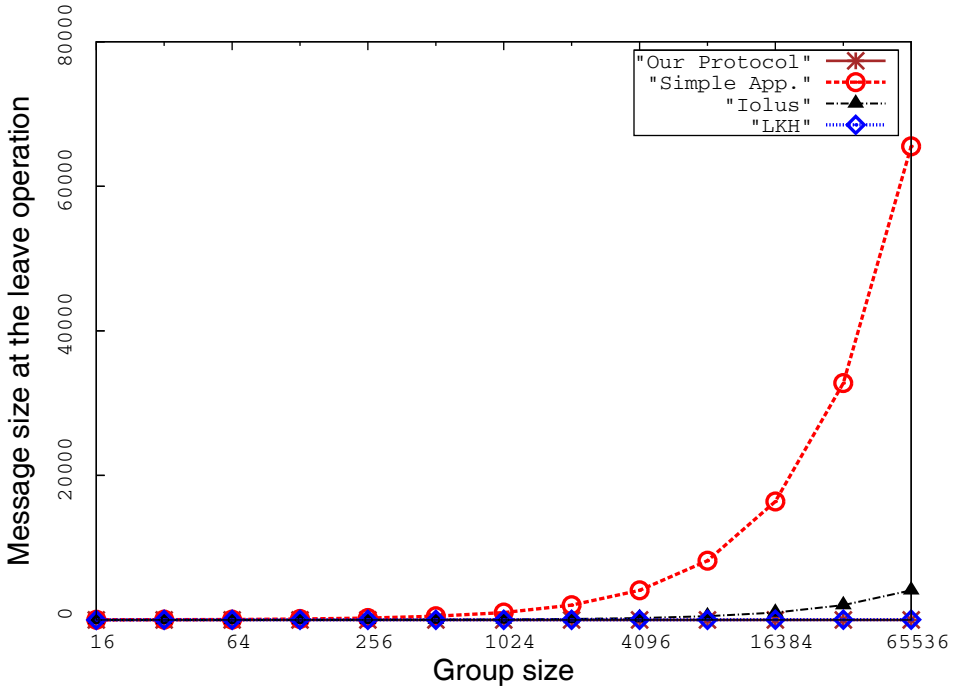


Figure 14: Message size overhead *at leave* operation

4.4. Storage overhead

Storage overhead is the number of keys stored at the key server, and by a member node.

In the simple approach, a member node stores only *its individual* key and the *group* key, but the key server stores *n* individual keys and the *group* key [24].

In LKH, the number of keys stored in the key server in *d-ary* protocols is equal to $\left\lceil \frac{d}{d-1} \right\rceil n$. So, the key server stores 2*n*, where *n* is the number of the group members. Each member stores all its path keys from itself to the root key which represents the group key in addition with its session key. So, each member stores $log_2 n + 1$ *keys* [2].

In the Iolus protocol, the subgroup controller of a subgroup of *m* members stores *m individual* keys and the subgroup key in addition with the group key shared with the other GSAs. On another hand, each member of a subgroup stores only its *individual* key and the *subgroup* key [9].

In the proposed protocol, the key storage in the key server is the sum of *2m-1* node keys, *m* inverse values of the subgroup members and the alternative key of the subgroup. So, the key server stores *3m* keys. Each member stores its path keys $log_2 m$ and *m-1* inverse values of the group members. So, each member stores $m + log_2 m - 1$.

	Key server	Member node
Simple App.	O(n)	0
LKH	O(n)	$O(\log_2 n)$
Iolus	O(m)	0
Our protocol	O(m)	O(m)

Table 5: Complexity of storage overhead at the key server and a member node

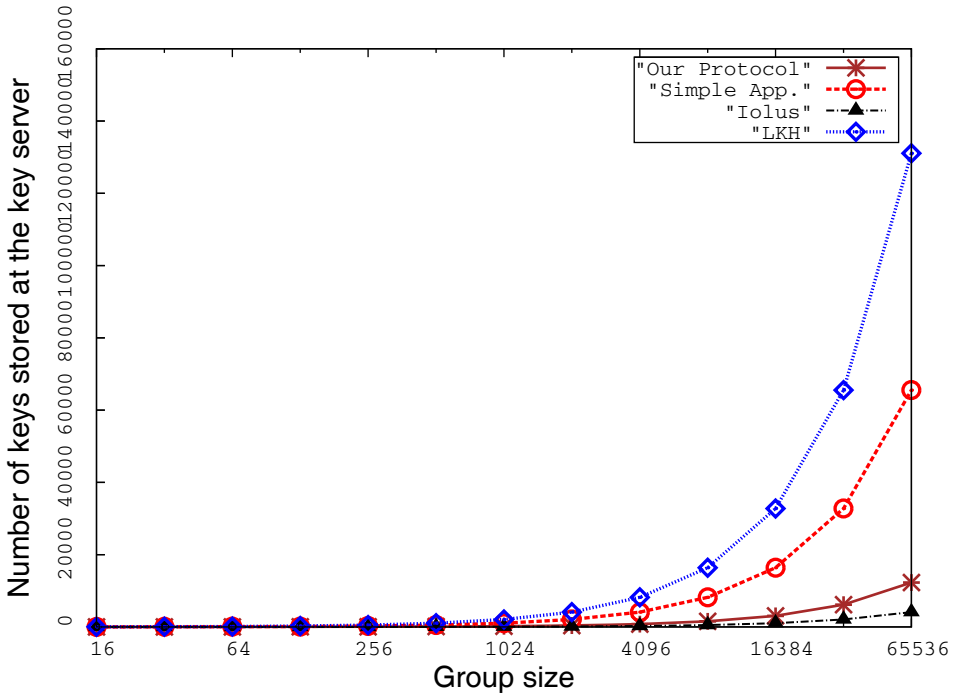


Figure 15: Storage overhead at the key server

Table 5 shows the time complexity of the storage overhead at the key server and a member node. Figures 15 shows the number of keys stored at the key server.

As shown in table 5 and figure 15, the Iolus protocol has the smallest storage overhead at both the key server and member nodes. Our protocol is worse at key storage overhead than the Simple App. and Iolus protocols at both the key server and member nodes but it is better than the LKH protocol as shown in figure 15.

From the above analysis, we can say that the proposed protocol enhances the group performance in terms of communication and computation overhead especially at the *leave* operation.

5. Conclusion

In this paper, we propose an efficient and a scalable protocol to solve the problem of distributing a symmetric key between the whole group members for secure multicast communication. It divides the whole group into several subgroups as in the Iolus protocol. Each subgroup in turn is organized in a logical key hierarchy as in the LKH protocol. This decomposition reduces the complexity for a member join or leave form $O(n)$ to $O(log_2 m)$, where n is the number of the whole group members and m is the number of each subgroup members. The performance of the proposed protocol is compared with that of the Simple App., Iolus and LKH protocols. The comparison is undertaken according to the computational overhead, communication overhead, storage overhead, and message size. The results show that the proposed protocol enhances the group performance in terms of computation overhead, and communication overhead especially at the leave operation.

References

1. K. C. Chan, S. -H. G. Chan, Key Management Approaches to Offer Data Confidentiality for Secure Multicast,IEEE Network, 17(5) 2003.
2. Chung KeiWong, Mohamed Gouda and Simon S. Lam, "Secure Group Communications Using Key Graphs", IEEE/ACM Transactions on Networking, 28(4), 2000,16-30.
3. Y. Challal, H. Seba, Group key management protocols: A novel taxonomy, International Journal of Information Technology 2(1)(2005) 105-118.
4. S. Rafaeli, D. Hutchinson, A survey of key management for secure group communication, ACM Computing Surveys 35(3), 2003, 309-329.
5. S. Setia, S. Zhu, SR. Jajodia, A scalable and reliable key distribution protocol for multicast group rekeying. Technical report. George Mason University, January 2002.
6. D.Wallner, E. Harder, R. Agree, Key Management for multicast: Issues and architectures, National Security Agency, RFC2627, June 1999.
7. Z. Jun, Z. Yo, M. Fanyuan, G. Dawu, B. Yingcai, An extension of secure group communication using key graph, Elsevier Information Sciences 176, 2006, 3060-3078.
8. W. H. D. Ng, M. Howarth, Z. Sun, H. Cruickshank, Dynamic Balanced Key Tree Management for Secure Multicast Communications, IEEE Transactions on Computers 56(5), 2007.
9. S. Mittra, Iolus: a framework for scalable secure multicasting, ACM SIGCOMM Computer Communication Review 27(4), 1997, 277-288.
10. S. Rafaeli, D. Hutchinson, Hydra : a decentralized group key management, Proceedings of the 11th IEEE International WETICE: Enterprise Security Workshop (June 2002), 62-67.
11. M. Peyravian, S. M. Matyas, N. Zunic, Decentralized group key management for secure multicast communications, Elsevier Computer Communications, 22, 1999,1183-1187.
12. Y. Kim, A. Perrig, G. Tsudik, Communication-efficient group key agreement,IFIP SEC'01 Conference, France, June 2001.
13. M. Steiner, G. Tsudik, M.Waidner, Key agreement in dynamic peer groups,IEEE Transactions on Parallel and Distributed Systems, August 2000.
14. X. Chan, B. N. W. Ma, C. Yang. M-CLIQUES: Modified CLIQUES key agreement for secure multicast,Elsevier Computers and Security 26, 2007, 238-245.
15. A. N. Pour, K. Kumekawa, T. Kato, S. Itoh, A hierarchical group key management scheme for secure multicast increasing efficiency of key distribution in leave operation, Elsevier Computer Networks 51, 2007, 4727-4743.
16. Heba K. Aslan, A scalable and distributed multicast security protocol using a subgroup-key hierarchy, Elsevier Computers and Security 23 2004, 320-329.
17. Douglas R. Stinson, cryptography: Theory and Practice, 3rd edition, Chapman and Hall/CRC Press, 2006.
18. Ralph Wittmann, Multicast Communication: Protocols, Programming, and Applications, Morgan Kaufmann Publishers Inc., May, 2000.
19. Sanjoy Paul, Multicasting on the Internet and its Applications, Springer, Jun, 1998).
20. Mustaque Ahamad, Multicast Communication in Distributed Systems, Ieee Computer Society Press Technology Series,Jan 1990.
21. Larry L. Peterson Morgan Kaufmann, Computer Networks: A Systems Approach, Fourth Edition, Morgan Kaufmann Publishers Inc., 4 edition ,March 2007.
22. Lawrence Harte, Introduction to Data Multicasting, IP Multicast Streaming for Audio and Video Media Distribution, Althos Publishing 2008.
23. Nicolas Bonmariage and Guy Leduc, A survey of optimal network congestion control for unicast and multicast transmission, Computer Networks 50(3), 2006,448-468.
24. Jiannong Cao, Lin Liao and Guojun Wang, Scalable key management for secure multicast communication in the mobile environment, Pervasive and Mobile Computing2(2), 2006,187-203.

Article received: 2009-06-18