A Key Recovery Mechanism for Reliable Group Key Management

Taenam Cho and Sang-Ho Lee

Dept. of Computer Science and Engineering, Ewha Womans University,
11-1 Daehyun-Dong, Seodaemun-Gu, Seoul 120-750, Korea
{tncho, shlee}@ewha.ac.kr

Abstract. Secret group communication can be achieved by encryption messages with a group key. Dynamic groups face the problem of changing the group key whenever members join or leave. One of the solutions to this problem is to send the updated group key to members via rekey messages in a secure manner. The recovery of lost keys consequently becomes important because a member cannot decrypt the group data if he loses these messages. Saving messages and resending them by KDC (Key Distribution Center) not only requires large saving space, but also causes the transmission and decryption of unnecessary keys. Furthermore, the keys in the unsaved messages cannot be recovered. This paper proposes an efficient method for recovering group keys. The group key generation method presented in this paper is simple, enabling us to recover group keys without storing and eliminating the transmission and decryption of useless auxiliary keys.

1 Introduction

One of the most important security issues in group communication is confidentiality. Confidentiality is necessary in limiting user access to data from distributed simulations or secret corporate conferences, as well as in maintaining billing information of commercial images and online Internet broadcast sources. To achieve confidentiality, only allowed users should gain access to data through encrypted communication using a shared group key[16]. Therefore, it requires safe sharing of the group key only among valid members and securing data against invalid members. Secure sharing of keys involves a KDC that distributes updated keys via rekey messages to members as other members join or leave. In such a system, the loss of rekey messages would disable the reception of the updated group keys, causing an inability to decrypt group data and possibly the failure of decryption of any follow-up rekey messages transmitted. Such reliability issues still remain to be solved[9][10]. Several methods to increase reception rates have been introduced[18][14][19], but none of them address the problem in the recovery of lost keys. Another research[11] proposed a scheme in which a member may calculate the lost keys using verification information. However, this proposal ignores the possibility of the member's loss of the verification information also, in which case he still cannot recover the lost keys. Therefore, more

J. Zhou, M. Yung, Y. Han (Eds.): ACNS 2003, LNCS 2846, pp. 372–386, 2003.
© Springer-Verlag Berlin Heidelberg 2003

research is required, especially on the problems caused by members' log-in/out and communication delays. This paper proposes a method based on a model in which a KDC generates and distributes keys to members, and also allows the recovery of lost keys when key updates are made after keys have been lost by communication delays or member's log-out.

There are 7 sections in this paper. Section 2 explains the requirements that a group key management must fulfill. Related studies are described in Section 3, and Section 4 proposes a key recovery mechanism. Security, reliability and efficiency analysis are found in Section 5. In section 6, the parameters for optimization are derived. Finally, section 7 presents the conclusion and suggestions for future research.

2 Requirements

The group key must be shared only with legitimate members, and those who are no longer legitimate members or are yet to join the group should not be allowed to access the group data. To achieve this, the following requirements must be met[11][5].

– GKS (Group Key Secrecy): guarantees that it is computationally infeasible for an adversary to discover any group key.
– FS (Forward Secrecy): guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover subsequent group keys.
– BS (Backward Secrecy): guarantees that a passive adversary who knows a subset of group keys cannot discover preceding group keys.

FS is ensure that a member cannot learn about new group keys after he leaves the group. BS is ensure that a joining member cannot learn about the previous group keys even after he joins. To fulfill these requirements, KDC must update the group key whenever a member joins or leaves the group. Group members are required to buffer any encrypted data and rekey messages they received until the encrypting keys arrive[18][14][19][15]. The legal members must be guaranteed that they can decrypt the data even if the data are transmitted while they are logged off. Our objective is not to recover all of the lost keys, but to recover the only the keys that are needed to decrypt the data and useful keys. Therefore, this paper defines the condition on key recovery for reliable group key management.

– KR (Key Recovery): legitimate members must be able to recover the lost group keys and useful auxiliary keys from KDC.

3 Related Studies

There are efficient solutions for scalable group key management in dynamic groups. Fiat and Noar suggested a solution that prevents the coalitions of more than k users from group data[3]. Mittra proposed Iolus[7] in which the group

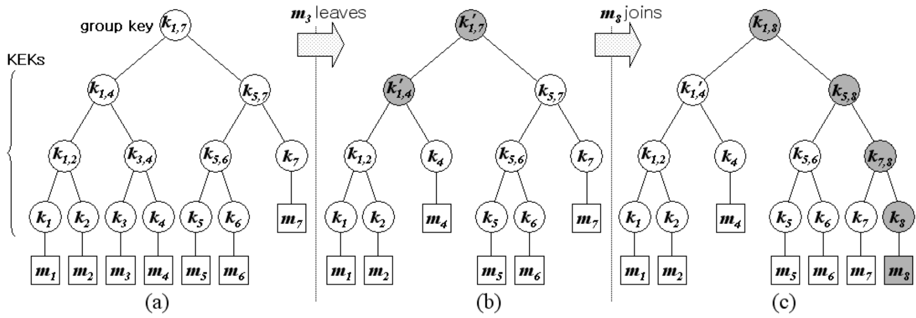

Fig. 1. A binary key-tree and key updates

is divided into several subgroups to be controlled independently. Another popular scheme, on which ours is based, is to employ a key-tree (LKH: Logical Key Hierarchy)[16][17]. LKH is secure against any number of corrupt users, and their rekey sizes are proportional to log$n$, where n is the group size, whereas Iolus requires a communication overhead proportional to the subgroup size. The improvement of the key-tree is based on the assumption that key-trees are balanced. The balance of the key-tree can be maintained by periodically updating[8] or by using AVL-tree[12]. The key values may be generated using a pseudo random number generator[17][8][12] or derived from child nodes' keys[11][1][13][2]. Since our method can be easily applied to other key-trees, the key-tree may be assumed to be in its simplest form, as a balanced binary tree. In this section, we introduce the binary key-tree with the studies for reliability.

3.1 Key Management Using a Key-Tree

Initially, KDC constructs a balanced binary key-tree whose leaf nodes nodes correspond to initial group members; the initial group may be empty. Each node of the tree is associated with a key. All members possess keys that are on the path from their leaf nodes to the root. The root key is the group key, and other keys are auxiliary keys; they will be called KEKs (Key Encryption Keys) hereafter (Refer to figure 1 (a)).

If m_3 in figure 1 (a) leaves, KDC updates the keys that m_3 possesses; KDC eliminates $k_3, k_{3,4}$ and replaces $k_{1,4}, k_{1,7}$ with $k'_{1,4}, k'_{1,7}$ (Refer to figure 1 (b)). Each new key is encrypted using its child keys and multicasted to other members; the rekey message may be such as $\{\{k'_{1,4}\}_{k_{1,2}}, \{k'_{1,4}\}_{k_4}, \{k'_{1,7}\}_{k'_{1,4}}, \{k'_{1,7}\}_{k_{5,7}}\}$. If m_8 joins in figure 1 (b), the key-tree is changed as shown in figure 1 (c). Each updated key is encrypted using the old key and the rekey message $\{\{k_{7,8}\}_{k_7}$, $\{k_{5,8}\}_{k_{5,7}}$, $\{k_{1,8}\}_{k_{1,7}}\}$ is multicasted. $\{M\}_{key}$ represents the encrypted message M of using a key, key. Therefore, the size of rekey message is proportional to the height of key-tree, $O(\log n)$.

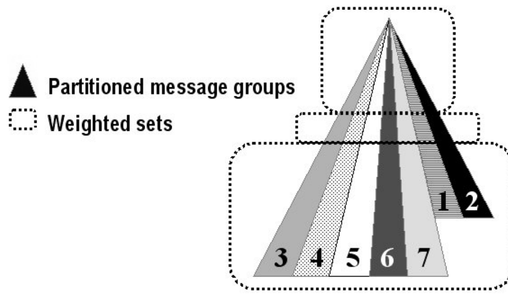

Fig. 2. Message composition

3.2 Group Key Management for Reliability

A group key distribution method with reliability was first proposed on Keystone[18]. This scheme proposes the utilization of FEC (Forward Error Correction) by the UDP transfer. [19], pointing out that the UDP transfer is incapable of dealing with burst errors, suggested a method of transferring rekey messages in divisions of many blocks. All keys needed by members are packed in a single block, and blocks are transmitted multiple times after RSE (Reed Solomon Error) codes are attached to each of them. [15] considers that keys on the upper levels of the key-tree are shared by a larger number of members, and it groups keys into several weighted classes. Keys of higher weights are entitled to more frequent transmissions. The 7 triangles in figure 2 indicate divided message blocks, and the 3 rectangles drawn in dotted lines indicate blocks with different weights. The three methods described above aim at better key update massage reception rates in soft real-time. This means that key loss cannot be completely prevented, and retransmissions are made repeatedly to prepare for key loss. However, these methods neglect the possibility of additional key updates taking place while previous retransmissions are delayed or not completed. Similarly, if a member is logged out, previous messages that should have been buffered may become lost. The two cases are the same in that the lost messages are not the last rekey messages.

In ELK[11], key verification information called *hint* is piggybacked on data for key recovery. The member who loses a rekey message message applies a brute force search for the lost keys. They can verify the candidate keys by checking them against the verification information. This method, invented to reduce data flow, necessitates large amounts of calculations by members. Nevertheless, if a member loses both the rekey message and the data, he can neither recover the lost keys nor decrypt the subsequent rekey messages. Therefore, a key recovery mechanism via KDC is necessary, not only for the recent rekey message but also for arbitrary past keys.

4 A Basic Key Recovery Mechanism

KDC maintains a binary key-tree as described in section 3.1 in our scheme. Whenever a member joins or leaves, KDC updates keys by executing procedure 1 or 2 respectively.

1. Authenticate the joining member, m_u, and send a new key, k_u, to m_u.
2. Create a leaf node n_u and a internal node n_p,
   and associate k_u and a new key k_p to them, respectively.
3. Let n_s and k_s be the shallowest leaf node in the current key tree, T_c, and its key.
   Let the corresponding keys from parent of n_s to the root be $< k_{i_1}, k_{i_2}, ..., k_{i_h} >$.
   Replace n_s with n_p, and attach n_s and n_u to n_p as child nodes.
4. Update $< k_{i_1}, k_{i_2}, ..., k_{i_h} >$ to $< k'_{i_1}, k'_{i_2}, k'_{i_h} >$
5. Encrypt $< k_p, k'_{i_1}, k'_{i_2}, k'_{i_h} >$ with k_u and unicast it to m_u.
   Encrypt k_p with k_s and k'_{i_j} with k_{i_j} $(1 \leq j \leq h)$, and multicast it to members.

Procedure 1. Join protocol

1. Let m_u and n_u be the leaving member and the corresponding leaf node
   in T_c, respectively.
   Let n_s and n_p be the sibling and the parent node of n_u, respectively.
   Let the corresponding keys from parent of n_p to the root be $< k_{i_1}, k_{i_2}, ..., k_{i_h} >$
2. Replace n_p with n_s, and remove n_u and n_p.
3. Update $< k_{i_1}, k_{i_2}, ..., k_{i_h} >$ to $< k'_{i_1}, k'_{i_2}, ..., k'_{i_h} >$
4. Encrypt each key of $< k'_{i_1}, k'_{i_2}, ..., k'_{i_h} >$ with its child node keys,
   and multicast it to remaining members.

Procedure 2. Leave protocol

A naive solution of recovering a message which is not the last rekey message is message retransmission; KDC sets the number of key recovery-supporting messages, w, as a system parameter, and prepares enough buffers in which messages of w can be stored. The appropriate value of w should be determined according to characteristics of applications. At key recovery request by a member, buffers are scanned for the requested message. The message, if it is found, is unicasted to the member. If the buffers are not holding the message, the member is notified of recovery inability. This method is simple but incomplete in that messages not stored in buffers cannot be recovered. If some of the lost KEKs are updated again before the lost rekey message is recovered, they become useless thereafter because they can no longer be used to decrypt other keys. In other words, the simple recovering of lost rekey message leads members to decrypt KEKs for which their uses have vanished. We further discuss this inefficiency in section 4.2 in detail. This section proposes a method that can perform efficient recovery at the loss of previous rekey message as well as the last message. This method also allows members to eliminate unnecessary key decryptions.

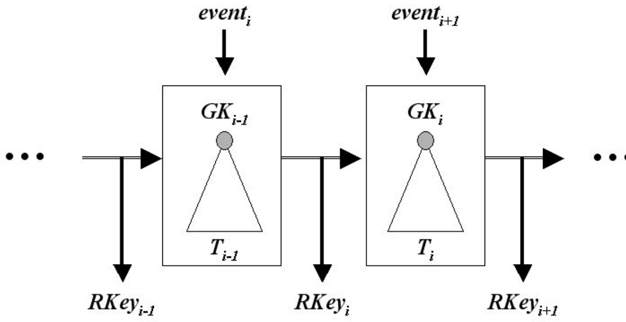

Fig. 3. Events, key-trees and rekey messages

4.1 Notations

In this section, we define the notations to explain our scheme with the following terms:

- $event_i$: An event which causes the ith key update. It is either the *join* or the *leave* of a member.
- T_i: The key-tree after the ith update from $event_i$ takes place (Refer to figure 3).
- $RKey_i$: The rekey message that contains keys updated from T_{i-1} to T_i (Refer to figure 3).
- GK_i: The root key of T_i which is the group key updated by $RKey_i$ in which case i is the version number of the group key (Refer to figure 1 and 3).
- m_i: The member whose identifier is i.
- k_i: The key shared only by the member m_i and KDC.
- $k_{i,j}$: The key shared by KDC and the members ranging from m_i to m_j when the key-tree is traversed by DFS (Depth First Search) (Refer to figure 1 and 3).
- $path_r^q$: The path from the leaf node of m_q to the root in T_r. For example, $path_r^1$ in figure 4 is $< k_1, k_{1,2}, k'_{1,4}, k'_{1,7} >$.
- R_r^q: The common path of $path_r^q$ and $path_r^x$ (Let the $event_r$ be caused by m_x). For example, R_r^1 in figure 4 is $< k'_{1,4}, k'_{1,7} >$.
- l_r^q: The level of the lowest node of R_r^q. For example, l_r^1 in figure 4 is the level of $k'_{1,4}$, 2 (Note that the root's level is 1 and level of a node is larger than that of the parent node by 1).
- $PRF_{key}(s)$: A pseudo random function that performs the mapping of s with the key, key[4].

This section refers to the member who has requested a key recovery as m_q, and to the lost rekey message and the group key as $RKey_r$r and GK_r, respectively. The group key, at the time of KDC receiving the key recovery request message from m_q, is GK_c.

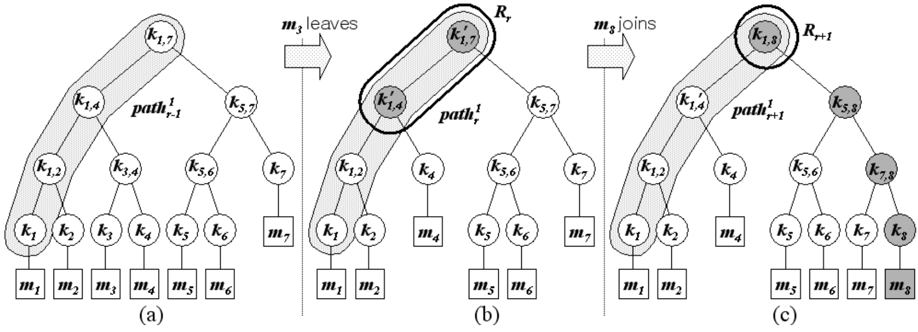

Fig. 4. Key Updates, T_i, $path_i^q$ and R_i^q

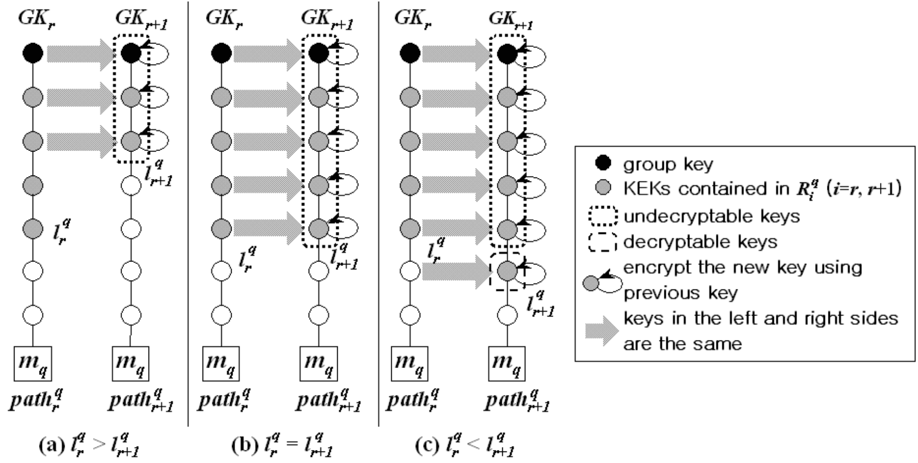

Fig. 5. In case of $event_{r+1} = join$

4.2 Observations

The following are observed from the analysis of the characteristics of LKH.

(1) If one key of the key-tree is updated due to $event_i$, then its ancestor keys are updated as well. That is, for arbitrary r and q, if we let the length of R_r^q be y, then R_r^q consists of the highest y keys of $path_r^q$ (Refer to section 4.1 and figure 4).

(2) Suppose m_q misses $RKey_r$. Then m_q cannot decrypt the following keys of $RKey_{r+1}$ due to the loss of $RKey_r$:

(a) In the case of $event_{r+1} = join$, m_q cannot decrypt the keys whose levels are less than or equal to l_r^q, since the keys used to encrypt those new keys are lost in $RKey_r$ (Refer to figure 5).
(b) In the case of $event_{r+1} = leave$ and $l_r^q > l_{r+1}^q$, m_q cannot decrypt any of the keys in R_{r+1}^q. The rekey message for R_{r+1}^q is constructed as a chain in

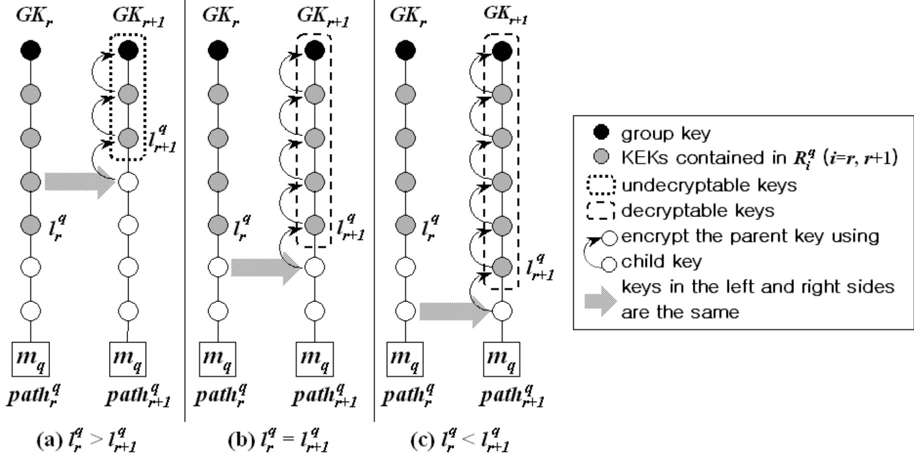

Fig. 6. In case of $event_{r+1} = leave$

which each key is encrypted using its child key. If $l_r^q > l_{r+1}^q$, the first key of the chain is lost in $RKey_r$ (Refer to figure 6 (a)). As the decryption of the chain must begin with the first key, anyone who does not know the first key cannot decrypt the keys in the chain.

(3) m_q can decrypt the following keys of $RKey_{r+1}$ without recovering $RKey_r$:

(a) In case of $event_{r+1} = join$, m_q can decrypt the nodes whose levels are greater than l_r^q, because the keys used to encrypt these keys are not lost in $RKey_r$ (Refer to figure 5 (c)).
(b) In the case of $event_{r+1} = leave$ and $l_r^q \leq l_{r+1}^q$, m_q can decrypt all the keys in R_{r+1}^q, because the first key of the chain is not lost in $RKey_r$ (Refer to figure 6 (b) and (c)). Anyone who knows the first key can decrypt all the keys in the chain.

(4) The first rekey message that m_q can decrypt without recovering $RKey_r$ is $RKey_s$. $\Longleftrightarrow$ s is the minimum value such that $l_r^q \leq l_s^q \wedge event_s = leave$.

(Proof) It can be proven by induction on s.
(a) In the case that the first index of rekey message that m_q can decrypt is $s = r + 1$:

If $event_{r+1} = leave \wedge l_r^q \leq l_{r+1}^q$, then m_q can decrypt R_{r+1}^q by (3)-(b); he can do so only if $event_{r+1} = leave \wedge l_r^q \leq l_{r+1}^q$ by (2).

(b) In the case that the first index of rekey message that m_q can decrypt is $s = r + i$, $i \geq 2$:

Since m_q cannot decrypt any of the keys of R_{r+1}^q, $event_{r+1} = join \vee l_r^q > l_{r+1}^q$ by (3). Consider the case of $event_{r+1} = join$ first. If $l_r^q > l_{r+1}^q$, the

levels of the undecryptable keys in figure 5 (a) are less than or equal to l_r^q; the two lowest keys in R_r^q are the same keys in $path_{r+1}^q$. In the case of $l_r^q \leq l_{r+1}^q$, the keys whose levels are less than or equal to l_r^q are not decryptable by (2)-(a). Similarly in the case of $event_{r+1} = leave \wedge l_r^q > l_{r+1}^q$, the keys whose levels are less than or equal to l_r^q are not decryptable. That is, m_q only does not know the keys whose levels are less than or equal to l_r^q on the $path_{r+1}^q$. The effect is the same as m_q losing R_{r+1}^q such that $l_r^q = l_{r+1}^q$. So, we can apply the same idea to j $(r < j < i)$ repeatedly. Consequently, if m_q can decrypt R_i^q, then $event_i = leave \wedge l_r^q \leq l_i^q$. And as R_i^q is the first decryptable rekey message, $event_j = join \vee l_r^q > l_j^q$ for $j(r < j < i)$.

(5) If m_q can decrypt $RKey_s(r < s)$ without recovering $RKey_r$, then he can decrypt all rekey messages thereafter.

(Proof) Since m_q can decrypt $RKey_s$, $event_s = leave \wedge l_r^q \leq l_s^q$ (Refer to (4)). As we described above, since the levels of the keys that m_q cannot decrypt are less than or equal to l_r^q, there are no undecryptable keys on the levels that are greater than l_r^q. That is, m_q knows every keys on $path_s^q$, therefore, he can decrypt all rekey messages thereafter.

(6) The keys that are not used to encrypt any other group keys or KEKs are not necessarily recovered or decrypted. In other words, if m_q can decrypt R_s^q and recover $GK_i(r \leq i < s)$, the KEKs of R_i^q $(r \leq i < s)$ whose levels are less than or equal to l_r^q are not useful for m_q.

4.3 Basic Key Recovery Algorithm

The basic idea of our scheme is searching s that satisfies the conditions started in section 4.2 (4), and then transmitting $GK_r, ..., GK_{s-1}$ that are the lost or undecryptable group keys, to m_q. In effect, KDC eliminates the transmission of useless KEKs, and the member who loses the rekey message has no need to decrypt those keys. The maximum size of R_r^q is $\log_2 n$, the height of the key-tree. Therefore, when $(s - r) > \log_2 n$, retransmitting R_r^q may be more efficient than our scheme. However, the expected value of $s - r$ is very small; we will analyze this value in section 5. Our scheme is as follows.

(1) Group Key Generation Method

KDC generates group keys in the following manner that any version of the group key can be generated at any time, while satisfying GKS, FS and BS. $MKey$ is a secret key which is only known to KDC, and the version of the group key is increased by 1 whenever the group key is updated beginning from 1.

$$GK_i = PRF_{MKey}(i).$$

(2) Node Structure of Key-tree

Each node of the key-tree has an array $flag[0, ..., w\text{-}1]$ and $last$ to calculate l_i^q. 2-bit $flag[i\%w]$ of each node records $event_i$ that causes an update of its key value. $last$ indicates the last index of an event that causes an update of the corresponding key. Whenever $event_i$ occurs by m_x, $flag[i\%w]$ of the nodes on $path_i^x$ is set to $event_i$, and $last$ is set to i. Note that $flag$s are used repeatedly with period w. Therefore, whenever key update and key recovery procedure are executed, past values of $flag$s must be cleared and the value of $last$ adjusted appropriately.

(3) Key Recovery Algorithm

KDC maintains the current key-tree, T_c only. Receiving the key recovery request for $RKey_r$ from m_q, KDC processes the procedure 3.

1. Check if $c - r + 1 \leq w$. Then,
    1.1 Calculate l_r^q.
        // This value is the lowest level of node of $path_c^q$
        // whose $flag[i\%w]$ is $join$ or $leave$.
    1.2 By using $flag$s, search for s that satisfies the conditions of observation (4).
    1.3 If such an s exists,
        1.3.1 Then, send $GK_r, ..., GK_{s-1}$ to m_q.
        1.3.2 Otherwise,
            // It means that there are no rekey messages decryptable
            // by m_q without recovering $RKey_r$. In this case, although
            // KDC should send R_i^q, it does not have any value of past keys.
            // KDC can use the current key-tree only.
            Send $GK_r, ..., GK_{c-1}$ and the keys of $path_c^q$ whose levels are
            less than or equal to l_r^q.
            // As the levels of undecryptable keys due to the loss of $RKey_r$
            // are less than or equal to l_r^q, m_q can decrypt R_c^q and GK_c.
2. Else, i.e., $c - r \geq w$, // KDC cannot calculate l_r^q.
    2.1 Send $GK_r, ..., GK_{c-1}$ and $path_c^q$ to m_q.
// The keys are unicasted to m_q after being encrypted using k_q.

Procedure 3. A basic key recovery algorithm

5 Analysis

5.1 Security and Reliability

Our rekey algorithm is the same as the previous algorithm[16] [17], except for the group key generation method. The group key is generated by PRF in our

scheme. PRF is calculated by using the secret key of KDC, $MKey$, as a key and the version number of the group key as a seed. Characteristics of the PRF[4] make it computationally infeasible to predict the value of the function or the key even if an attacker knows a set of group keys. Therefore, FS and BS can be guaranteed because even a member who possesses certain parts of the group keys is unable to calculate the preceding/succeeding keys or $MKey$. These properties can be provided using HMAC[6] as a PRF. Another security point is in the key recovery procedure. We assume that KDC maintains members' information, such as the subscription period, for membership management. Using this, KDC can check the validity of members' key recovery requests. An adversary may try to obtain group keys by guessing the group key or $MKey$, or by eavesdropping on the key recovery messages. Because of the property of PRF, an adversary cannot discover $MKey$ or group keys by guessing even though the version number of group key is public. Thus, GKS is guaranteed if the encryption algorithm is secure; note that the recovered keys are encrypted using the individual key of the member.

KDC is capable of recovering arbitrary group keys at members' requests because group keys can be calculated only from $MKey$ and the group key version in our scheme. KDC also calculates group keys and KEKs that cannot be decrypted without the recovery of the lost keys. Our method, therefore, provides reliability with successful key recovery.

5.2 Efficiency

(1) Additional Storage on KDC

Let K be the bit length of a key. In the naive solution, KDC needs additional storage of approximately $2wK{\cdot}\log n$ bits to buffer past rekey messages. The required storage is increased to about $2wn$ bits in our scheme since each node of the tree has to have a $flag$. However, this may be reduced optimally. The optimization will be described in section 6.

(2) Traffic Amount and the Number of Encryptions/Decryptions

The loads on KDC and members are estimated by the number of key encryptions/decryptions and the amount of traffic is measured by the number of transmitted keys. For the analysis of the average and the worst case, let $p = c - r + 1$ and $t = s - r$; p indicates that how many key updates occur after the loss of $RKey_r$; t indicates the number of undecryptable rekey messages due to the loss of $RKey_r$. Let the average size of R_i^j be u.

In the case that the lost rekey message is not within the recovery window w, our scheme still can recover the lost message that the naive solution cannot. Such recovery would require as much work load as would the worst case. Otherwise, in the naive solution, $\log_2 n$ or $2{\cdot}\log_2 n$ keys are transmitted for join or leave events, respectively. Thus, the average number of keys to be transmitted is $1.5{\cdot}\log_2 n$. Because m_q cannot decrypt $RKey_r, ..., RKey_c$ in the worst case, he may decrypt $\log_2 n$ keys for each $RKey_i$ after recovering the $RKey_r$. In average, he may

Table 1. The number of computations and transmissions

Cases	Methods (Subjects / Criteria)		KDC (The number of encryptions)	Traffic (The number of keys)	Member (The number of decryptions)
$r \le c - w$	Naive solution		-	-	N.A.
	Proposed scheme		$p - 1 + \log_2 n$	$p - 1 + \log_2 n$	$p - 1 + \log_2 n$
$c - w < r \le c$	Worst case	Naive solution	0	$2 \cdot \log_2 n$	$p \cdot \log_2 n$
		Proposed scheme	$p - 1 + \log_2 n$	$p - 1 + \log_2 n$	$p - 1 + \log_2 n$
	Average	Naive solution	0	$1.5 \cdot \log_2 n$	$u \cdot t$
		Proposed scheme	t	t	t

decrypt u keys for each $RKey_i (r \le i < s)$. In our scheme, KDC transmits t keys, $RKey_r, ..., RKey_{s-1}$. In the worst case, when $t = p$, KDC transmits $p - 1$ group keys, $GK_r, ..., GK_{c-1}$ and $\log_2 n$ keys of $path_c^q$. The number of encryptions of KDC and the number of decryptions of a member is equal to the number of transmitted keys.

Comparisons between the proposed method and the naive solution are made on Table 1. It shows that our scheme reduces the traffic amount and process load on members instead of the load on KDC, except for the traffic amount in the worst case. However, the practical efficiency of schemes should be estimated not by its performance in the worst case but by that on average. In section 6, we will derive the expected value of t.

6 Optimization and Analysis on Expectation

As described in section 5.2, the storage of KDC in our scheme is proportional to the group size. It is not scalable for large groups. To eliminate this constraint, we try to attach $flag$s only to the highest nodes of the key-tree instead of attaching it to all nodes. In this section, we derive the appropriate level of the node to which to attach $flag$ and analyze u and t for average efficiency.

6.1 Space Optimization

For simplicity, we assume that $n = 2^h$ for some h. Let the keys of $path_r^x$, ordered by from the root to the leaf, be $k_{i_1}, k_{i_2}, ..., k_{i_h}$. Assume also that $event_r$ occurs with the join or leave of m_x. Without loss of generality, let m_x be the rightmost member in the key-tree. Then as shown in figure 7, the leftmost $n/2$ members need only the root key k_{i_1}. Next $n/2^2$ members need 2 keys, k_{i_1} and k_{i_2}. The total number of key updates for group members, except for m_x, is $\sum_{i=1}^{h}(2^{-i} \cdot n \cdot i) = \left(2 - \frac{\log_2 n + 2}{n}\right) \cdot n$. If the probability of key losses for all members is uniform, the expected number of keys to be recovered, u, is $2 - \frac{\log_2 n + 2}{n} \approx 2$.

Using this fact, we can reduce the amount of additional storage needed by KDC. If we maintain $flag$s on only the nodes whose levels are less than or

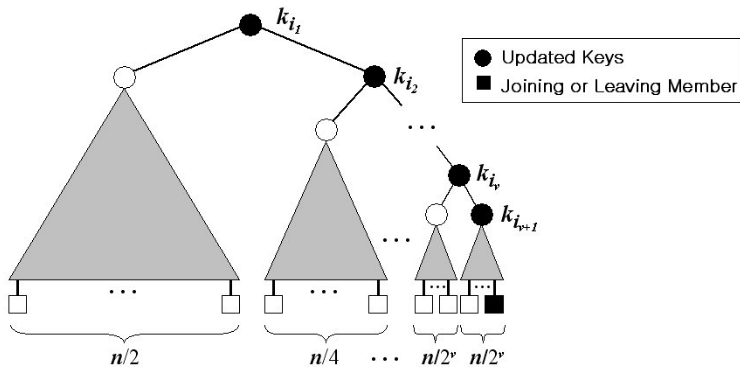

Fig. 7. The number of related members for each key

equal to v, the amount of storage is about $2w \cdot (2^v - 1)$ bits. If $l_r^q \geq v$ for some r and q, as all $flag$s of $path_r^q$ are set to $event_r$, KDC cannot know the real l_r^q. In this case, we regard l_r^q as $\log_2 h$. Therefore, the expected number of keys to be recovered converges to 2 as follows:

$$\frac{1}{n}\left\{\sum_{i=1}^{n}(2^{-i}\cdot n\cdot i)+(1-\sum_{i=1}^{n}2^{-i})\cdot n\cdot h-\frac{1}{n}\cdot h\right\}=2+\frac{\log_2 n-v-2}{2^v}-\frac{\log_2 n}{n}.$$

For various v and n, the expected number of keys to be recovered is shown in table 2. If we set v to 4 when n is not greater than 1 million, and to 5 otherwise, we can maintain the number of keys to be recovered below 3. As a result, the additional storage for KDC is reduced to $62w$ bits regardless of the group size.

6.2 Expectation of the Amounts of Traffic

Now, we analyze the expected value of t which is a major factor of efficiency in our scheme. We assume that the probabilities of the occurrence of a member join and that of a leave are equal. The probability of $l_r^q = i$ is 2^{-i}. For each i, the probabilities of $l_r^q < l_{r+1}^q$, $l_r^q = l_{r+1}^q$ and $l_r^q > l_{r+1}^q$ are 2^{-i}, 2^{-i} and $1 - 2^{-i+1}$,

Table 2. Key recovery message size, u according to v

v \ n	100	1,000	10,000	1.E+05	1.E+06	1.E+07	1.E+08	1.E+09	1.E+10
2	2.59	3.48	4.32	5.15	5.98	6.81	7.64	8.47	9.30
3	2.14	2.61	3.03	3.45	3.87	4.28	4.70	5.11	5.53
4	1.97	2.24	2.45	2.66	2.87	3.08	3.29	3.49	3.70
5	1.92	2.08	2.20	2.30	2.40	2.51	2.61	2.72	2.82
6	1.91	2.02	2.08	2.13	2.19	2.24	2.29	2.34	2.39
7	1.92	2.00	2.03	2.06	2.09	2.11	2.14	2.16	2.19
8	-	1.99	2.01	2.03	2.04	2.05	2.06	2.08	2.09
$\log_2 n$	6.64	9.97	13.29	16.61	19.93	23.25	26.58	29.90	33.22

Table 3. Expectations of computations and traffic amount

Methods \ Subjects (Criteria)	KDC (The number of encryptions)	Traffic (The number of keys)	Member (The number of decryptions)
Naive Solution	0	$1.5 \cdot \log_2 n$	6 (9)
Proposed Scheme	3	3	3

respectively. Consequently, for an arbitrary l_r^q, the probabilities of $l_r^q < l_{r+1}^q$, $l_r^q = l_{r+1}^q$ and $l_r^q > l_{r+1}^q$ are $\sum_{i=1}^{h-1}\left(2^{-2i}\right) = \frac{1}{3}\left(1 - \frac{4}{n^2}\right)$, $\sum_{i=1}^{h}\left(2^{-2i}\right) = \frac{1}{3}\left(1 - \frac{1}{n^2}\right)$ and $\sum_{i=1}^{h}\left(2^{-i} \cdot (1 - 2^{-i+1})\right) = \frac{1}{3}\left(1 - \frac{9n-6}{n^2}\right)$, respectively. Since the probability of $s = r+1$ is equal to that of $(l_r^q \leq l_{r+1}^q \wedge event_{r+1} = leave)$, it is about 1/3. The probability of $s = r+2$ is equal to that of $(l_r^q > l_{r+1}^q \vee event_{r+1} = join) \wedge (l_r^q \leq l_{r+2}^q \wedge event_{r+2} = leave)$, and it is about 2/9. In general, the probability of $s = r+t$ is about $2^{t-1}/3^t$, so the expectation of t is $\lim_{t\to\infty} \sum_{i=1}^{t}\left(t \cdot 2^{i-1} \cdot 3^{-i}\right) \cong 3$.

We implemented LKH with the balancing method proposed in [8]. The result of this experimentation shows that u converges to 3. Based on this result, the average of computations and the traffic amount are presented in table 3. As shown in table 3, the number of computation and the traffic amount of our scheme are constants, and the number of decryptions is reduced to 1/3 of the decryptions required by the naive solution in maximum.

7 Conclusion and Future Work

This paper analyzed and defined problems related to rekey message loss and to members' log-out. Based on this analysis, we proposed a key recovery mechanism that recovers arbitrary group keys efficiently and eliminates transmission and decryptions of useless KEKs without storing any rekey messages. Our scheme reduced the average traffic amount and the number of decryptions for members at the cost of encryption overheads at KDC from those of the naive solution. Our scheme is based on binary a key-tree, which is the simplest form of LKH. Other schemes such as [1] and [2] are designed to improve the efficiency of LKH. It is expected that our scheme can be applied to those schemes with slight modification. However, more exact analysis on the efficiency is necessary.

The proposed method is triggered by key recovery requests from members to KDC. Keys are more likely to be lost in cases of unstable networks. Such cases result in an increase in recovery requests, congesting data transmissions to KDC. Combinations created with methods of [18], [14] or [19] could prevent the problem. These methods alone would require repeated transmission with reception percentages close to 100%. Therefore, recovering the lost messages by the proposed method, after the application of the method of repeated transmission to guarantee appropriate degrees of reception rates, would be more efficient. Research on proper reception rates are needed in the future to maximize the efficiency.

References

1. D. Balenson, D. McGrew and A. Sherman, “Key Management for Large Dynamic Groups: One-way Function Trees and Amortized Initialization,” IETF Internet Draft, 1999.
2. R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor and B. Pinkas, ”Multicast Security: A Taxonomy and Some Efficient constructions,” Proc. of IEEE INFOCOMM’99, Vol. 2, pp708–716, 1999.
3. A. Fiat and M. Naor, ”Broadcast Encryption,” Advances in Cryptology: Proc. of Crypto’93, LNCS 773, pp480–491, 1994.
4. O. Goldreich, S. Goldwasser and S. Micali, “How to Construct Random Functions,” Journal of the ACM, Vol.83, Issue 4, pp792–807, 1986.
5. T. Hardjono, B. Cain and B. Doraswamy, “A Framework for Group Key Management for Multicast Security,” IETF Internet Draft, 2001.
6. H. Krawczyk, M. Bellare and R. Canetti, “HMAC: Keyed-hashing for Message Authentication,” IETF RFC 2104, 1997.
7. S. Mittra, ”Iolus: A Framework for Scalable Secure Multicasting,” Proc. of ACM SIGCOMM’97, Vol.27, Issue 4, pp277–288, 1997.
8. M. J. Moyer, J. R. Rao and P. Rohatgi, “Maintaining Balanced Key Trees for Secure Multicast,” IRTF Internet Draft, 1999.
9. http://www.ietf.org/html.charters/msec-charter.html
10. http://www.securemulticast.org/msec-index.htm
11. A. Perrig, D. Song and J. D. Tygar, “ELK, a New Protocol for Efficient Large-Group Key Distribution,” 2001 IEEE Symposium on Security and Privacy, pp247–262, 2001.
12. O. Rodeh, K. P. Birman and D. Dolev, “Optimized Group Rekey for Group Communication Systems,” Network and Distributed Systems Security 2000, pp37–48, 2000.
13. S. Rafaeli, L. Mathy and D. Hutchison, “EHBT: An Efficient Protocol for Group Key Management,” 3rd International Workshop on Networked Group Communications, pp159–171, 2001.
14. S. Setia, S. Zhu and S. Jajodia, “A Scalable and Reliable Key Distribution Protocol for Group Rekeying,” Technical Report, George Mason Univ., 2002.
15. S. Setia, S. Zhu and S. Jajodia, “A Comparative Performance Analysis of Reliable Group Rekey Transport Protocols for Secure Multicast,” Proc. of the Performance 2002 Conference, 2002.
16. C. K. Wong, M. Gouda and S. S. Lam, “Secure Group Communications using Key Graphs,” Proc. of ACM SIGCOMM, Vol 28, Issue 4, pp68–79, 1988.
17. D. Wallner, E. Harder and R. Agee, “Key Management for Multicast: Issues and Architectures,” IETF RFC 2627, 1999.
18. C. K. Wong and S. Lam, “Keystone: A Group Key Management Service,” Proc. of International Conference on Telecommunications, 2000.
19. X. B. Zhang, S. S. Lam, D. Lee and Y. R. Yang, “Protocol Design for Scalable and Reliable Group Rekeying,” Proc. of SPIE Conference on Scalability and Traffic Control in IP Networks, 2001.