www.redpel.com
+917620593389

# Efficient Rekeying Framework for Secure Multicast with Diverse-Subscription-Period Mobile Users

Young-Hoon Park, Dong-Hyun Je, Min-Ho Park†, and Seung-Woo Seo† *Member, IEEE,*

**Abstract**—Group key management (GKM) in mobile communication is important to enable access control for a group of users. A major issue in GKM is how to minimize the communication cost for group rekeying. To design the optimal GKM, researchers have assumed that all group members have the same leaving probabilities and that the tree is balanced and complete to simplify analysis. In the real mobile computing environment, however, these assumptions are impractical and may lead to a large gap between the impractical analysis and the measurement in real-life situations, thus allowing for GKM schemes to incorporate only a specific number of users. In this paper, we propose a new GKM framework supporting more general cases that do not require these assumptions. Our framework consists of two algorithms: one for initial construction of a basic key-tree and another for optimizing the key-tree after membership changes. The first algorithm enables the framework to generate an optimal key-tree that reflects the characteristics of users' leaving probabilities, and the second algorithm allows continual maintenance of communication with less overhead in group rekeying. Through simulations, we show that our GKM framework outperforms the previous one which is known to be the best balanced and complete structure.

**Index Terms**—multicast, security, group key, group key management, logical key hierarchy, batch rekeying, group dynamics

## 1 Introduction

In next-generation mobile communication, multicast service will be a key application for supporting a large group of subscribers simultaneously. Because multicast transmits data to the group simultaneously, it reduces the communication cost significantly [1][2]. However, since multicast may be vulnerable to an overhearing attack, this efficiency can be achieved only when security or access control is guaranteed; i.e., only authorized group members can read the data properly. Many commercial applications such as PayTV, vehicular ad hoc network (VANET), and group signature require that only legal users have access authority [3], [4]. Nowadays, because smart phones are becoming popular, many mobile applications which need group communication such as DMB, video conference, and online game have emerged.

For both security and efficiency, a group key (GK) which is shared only by a group of mobile devices has been employed for access control. A message for the group is encrypted by the GK and transmitted only once. Then the transmitted message can be decrypted by only group members having the GK. However, the GK is updated whenever the group membership changes for forward and backward secrecy, which can cause a serious problem with rekeying overhead [5].[1] Hence, many researchers have proposed variations of group key management(GKM), and have attempted to reduce the overhead for group rekeying.

Recently, as the number of smart phones grows, many various membership-based applications are developing. Many of these such as charged video streaming, online game and wireless access applications rely on paid service. Due to the increase of short-period of subscription, the group key is updated frequently, so that the communication overhead from rekeying grows. Hence, the efficient GKM scheme which can manage large and dynamic user group is necessary.

### 1.1 Related Works

For GKM, one of the most important concerns is how to minimize the communication overhead when the shared GK is updated among the subscribers [10], [11]. Tree-based GKM, one class of GKM, has received considerable attention from many researchers since the amount of overhead for group rekeying is proportional to the logarithm of the group size [12], [13], [14], [15], [16],

*†: Corresponding author*

*This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2012-0000918), and by the Institute of New Media and Communications, Seoul National University.*

*Y.H.Park is with Cloud Solution Team of DMC R&D Center in Samsung Electronics, e-mail: yhpark@cnslab.snu.ac.kr*
*D.H.Je is a senior engineer of Telecommunication-Systems Business in Samsung Electronics, e-mail: jdh317@gmail.com*
*M.H.Park is with School of Electronic Engineering, Soongsil University, Seoul, Korea, email: mhp@ssu.ac.kr*
*S.W.Seo is with the Department of Electrical Engineering, Seoul National University, Seoul, South Korea, e-mail: sseo@snu.ac.kr*

1. The rekeying overhead includes the communication overhead from the distribution a new GK to the remaining group members, the storage overhead from the saving of keys, and the computational overhead from en/decryption. Since the wireless bandwidth is much more expensive resource than the others in the next-generation mobile communication, we focus the communication overhead which is a dominant factor, similarly to previous work [6][7][8][9].

www.redpel.com
+917620593389

[17]. In addition, many researchers have proposed new GKM schemes with tree-based structure such as *logical key hierarchy*(LKH), [10], [18], *one-way function tree*(OFT) [19], and *one-way key derivation*(OKD) [7].

The most prominent tree-based GKM is LKH, which was independently proposed by Wong *et al.* [10] and Wallner *et al.* [18]. To accurately estimate and further reduce the communication overhead, many researchers have attempted to prove the lower bound of the communication cost and accurately analyze the communication cost. For an $a$-degree LKH, it has been proven that the communication overhead for each group rekeying is $O(a \log_a N)$, where $N$ is the group size. Zhu demonstrated that an LKH with a 4-degree tree structure is optimal [8]. Canetti *et al.* and Snoeyink *et al.* determined the lower bound of the communication cost [20], [5]. In addition, Micciancio *et al.* [21] provided tighter analysis on lower bound that matches the upper bound of [14].

Although many variations on LKH have been proposed to reduce the cost for group rekeying, they suffer from inefficient rekeying and out-of-sync problems. To solve these problems, Li *et al.* proposed batch rekeying(BR) [22]. In BR, rekeying occurs periodically, whereas in LKH, it occurs whenever the group membership changes. Although BR cannot guarantee the perfect forward and backward secrecy, in most of the pragmatic applications, some security leakage does not matter. In PayTV service, for example, it is more efficient that group rekeying is processed everyday than that it is processed whenever group members change.

Many researchers have investigated the exact communication cost of a key-tree in an attempt to reduce the cost of group rekeying. Some, including Li *et al.*, Zhu *et al.* and Lin *et al.*, have analyzed a balanced and complete key-tree [22], [8], [7]. Furthermore, Lee *et al.* and Je *et al.* obtained a more accurate communication cost model of a key-tree other than the balanced and complete key-tree, i.e., a *level-homogeneous key-tree* [9], [23].

The optimality of a key-tree has also been investigated, and it was shown that optimality can be achieved from the balance of a key-tree [24], [25], [26]. It is known that for the worst case of an unbalanced key-tree, the communication cost may be $O(N)$, where $N$ is the group size. In [24], [25], and [26], algorithms were proposed for generating a balanced tree structure. However, the performance of a key-tree may degrade when the locations of leaf nodes mapped to individual keys(IKs) for leaving users are widely distributed. To solve the worst-case problem, Ji *et al.* proposed a departure dependent key topology(DDKT)[27].

### 1.2 Motivation

Although many GKM proposals have been made since BR was proposed, they have some fundamental limitations to be adopted to the mobile environment, as follows.

1) They assumed the special case that all members have the same leaving probability, which is an impractical assumption. In the mobile situations, however, there are various types of members, i.e., from short-period to long-period subscribers. Although the impact of this unrealistic assumption on the results is quite significant, as this paper will show, the assumption has been broadly adopted by many researchers.
2) Because of the assumption in 1), most previous mathematical results that proved that GKM with a balanced and complete key-tree structure can achieve optimal performance may have limited application. Hence, the initial construction algorithms proposed for generating a balanced and complete key-tree structure that is known to achieve the best rekeying performance may also have limited applicability [7], [22].
3) Owing to the result in 2), key-tree optimization algorithms that maintain as balanced and complete a tree structure as possible after group membership changes (joining or leaving) cannot achieve optimal performance in reality [8], [24], [25], [26].
4) Owing to the result in 2), performance analysis was possible only for $a^d$, which is the number of members filling in the balanced and complete tree, where $a$ and $d$ are the degree and the depth of the tree structure, respectively. Therefore, the analysis results and algorithms are not applicable for an arbitrary number of mobile subscribers.

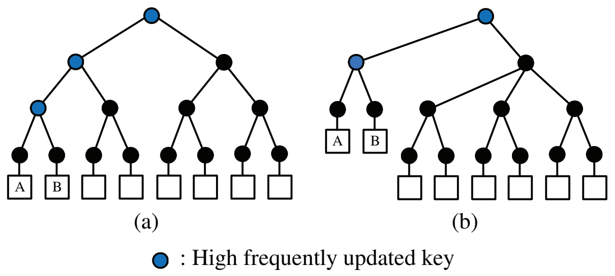


Fig. 1. Counterexample of the results in previous works

Nevertheless, the assumption of a special case has been accepted as applicable to general mobile environments, since no generic analysis methods for arbitrary leaving probabilities have been available. Fig 1 is an example demonstrating the falseness of this belief directly. The balanced and complete tree shown in Fig. 1 (a) has been considered as the optimal structure in the previous works. However, it is not true in most cases where members have different leaving probabilities. Consider the case that two member A and B is more likely to leave the group, i.e., they have higher leaving probabilities than the others. Three keys colored with blue in Fig. 1 (a) will be updated more frequently. In this case, the key tree structure as shown in Fig. 1 (b) performs better since there are just two blue-colored keys to be updated.

www.redpel.com
+917620593389

| | Previous works | Our approach |
|---|---|---|
| Leaving probability | equal | not equal |
| Initial key-tree | balanced & complete | general |
| After group membership changes | balanced & complete key-tree must be maintained | general key-tree is allowed |
| Number of users | specific($a^d$) | any number |

TABLE 1
Comparison to the previous works

### 1.3 Contributions and Organization

The main contribution of this paper is the development of a new framework for optimal GKM with dynamic mobile subscribers. The proposed framework consists of cost-efficient key-tree generation and management. We also provide a new mathematical analysis methodology for quantifying the performance of key-trees. The contributions of this paper can be summarized as follows:

1) We propose a new mathematical analysis methodology that can provide the precise average value of communication overhead for group key updates under general conditions. The conditions include an arbitrary number of members, non-equal leaving probabilities, and non-balanced and non-complete tree structure can support the mobile situations. Note that unlike previous works, the average size of rekeying messages can be calculated even though the subscription periods are diverse. Also, through our analysis, we find the conditions for the optimal tree structure that minimizes communication overhead.
2) We develop a two-step mechanism for optimal key-tree generation: one for initial key-tree generation followed by key-tree maintenance after the group membership changes. The first algorithm can generate a key-tree that corresponds to the optimal key-tree obtained by mathematical analysis.
3) For the second step of the mechanism in 2), we propose an optimal key-tree maintenance algorithm for use after the group membership changes. The algorithm optimizes the key-tree by modifying the tree structure considering the diverse-subscription periods of the mobile users.

In order to emphasize the contribution, we show the comparison between previous work and our approach as shown in Table1. While previous work is limited to some specific cases, our work is applicable to practical and general cases.

The remainder of this paper is organized as follows. Section 2 describes the model of our framework. In Section 3, we analyze the communication overhead of the key-tree by calculating the average size of rekeying messages. The properties of the optimal key-tree are discussed with certain theorems and corollaries in Section 4. In that section, we propose the methodology for generating the key-tree and locating the keys on the key-tree to minimize the average size derived in Section 3. After that, we propose a GKM framework with two algorithms for initial construction and optimization of the key tree is proposed in Section 5 and the simulation result is shown in section 6. Finally, we conclude this paper in section 7.

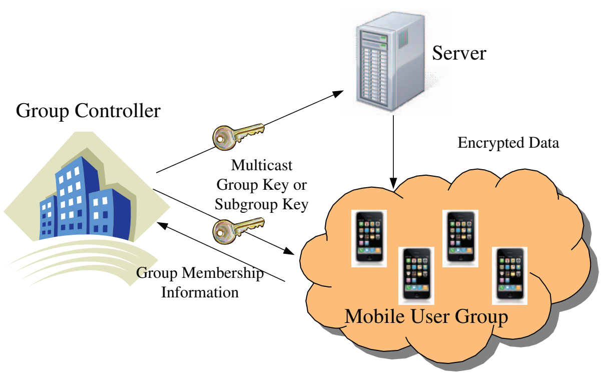


Fig. 2. Model of multicast environment

## 2 Model of Framework

The model of the proposed framework consists of a *group controller*(GC), a *server*, and a *mobile user group*. The GC manages all key types that are used to encrypt data or other keys, and transmits the keys that are updated upon changes in the mobile user group. A server sends group members data that is encrypted with the group key. Figure 2 shows a model of a multicast environment.

The *traffic encryption key*(TEK), which is shared by the server and all users in the group, is used to encrypt data, so the server encrypts the data that is sent to users only once. For forward and backward secrecy, the TEK should be updated whenever the group membership changes.

We adopt BR to achieve efficient rekeying in an environment where users join and leave frequently. A tree structure in BR has two types of nodes. Let **T** be a key-tree. There is a bijection mapping between $\mathcal{U}_\mathbf{T}$ and the set of user-nodes, and between $\mathcal{K}_\mathbf{T}$ and the set of key-nodes, where $\mathcal{U}_\mathbf{T}$ and $\mathcal{K}_\mathbf{T}$ are the sets of all users and all keys in **T**, respectively. User-nodes are located on the leaves of **T**, and key-nodes are located on the other nodes. On the parent node of each user-node, an IK is placed, which is shared by only the key server and the corresponding user. The TEK is located on the root of the tree, and is shared by the key server and all users in $\mathcal{U}_\mathbf{T}$. Finally, the *key encryption keys*(KEKs) are located on the remaining nodes, which are shared by users whose user-nodes are descendents of a key-node for the KEK.

Let $\mathcal{R}_\mathbf{T}$ be a set of relationships $(u, k)$ such that key $k$ is shared to user $u$. Then, $(u, k) \in \mathcal{R}_\mathbf{T}$ if and only if the key-node of $k$ is an ancestor of the user-node of $u$. As an example, the secure group of the key-tree in Fig 3 is

www.redpel.com
+917620593389

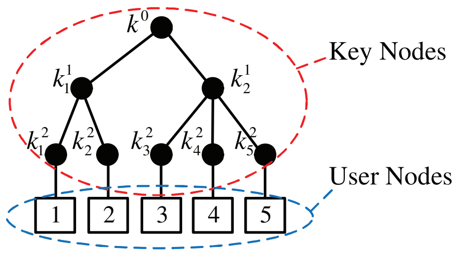


Fig. 3. User-nodes and key-nodes in key-tree.

as follows:

$$\begin{aligned}\mathcal{U}_{\mathbf{T}} &= \{u_1, u_2, u_3, u_4, u_5\},\\ \mathcal{K}_{\mathbf{T}} &= \{k_1^2, k_2^2, k_3^2, k_4^2, k_5^2, k_1^1, k_2^1, k^0\},\\ \mathcal{R}_{\mathbf{T}} &= \{(u_1, k_1^2), (u_2, k_2^2), (u_3, k_3^2), (u_4, k_4^2), (u_5, k_5^2),\\ &\quad (u_1, k_1^1), (u_2, k_1^1), (u_3, k_2^1), (u_4, k_2^1), (u_5, k_2^1),\\ &\quad (u_1, k^0), (u_2, k^0), (u_3, k^0), (u_4, k^0), (u_5, k^0)\}.\end{aligned}$$

Also, we define the following sets.

$$\mathcal{U}_{\mathbf{T}}(k) = \{u|(u,k) \in \mathcal{R}_{\mathbf{T}}\},\ \ \mathcal{K}_{\mathbf{T}}(u) = \{k|(u,k) \in \mathcal{R}_{\mathbf{T}}\}$$

Note that $\mathcal{U}_{\mathbf{T}}(k)$ is a set of users who shares key $k$, and $\mathcal{K}_{\mathbf{T}}(u)$ is a set of keys that user $u$ has.

When the group of users changes, the TEK and some KEKs should be updated. In BR, key updates are processed periodically in a batch mode. The message that contains updated keys is called a *rekeying message*, and the interval between consecutive key updates is called *Rekey Interval*. For example, let us consider user 3 and user 4 leave the group in a rekey interval. All the keys that they had possessed, $k_2^1$, $k^0$, should be updated. Some messages, e.g., $\{k(new)^0\}_{k_1^1}$, $\{k(new)^0\}_{k(new)_2^1}$, $\{k(new)_2^1\}_{k_5^2}$, are transmitted to update the old keys, which are the rekeying messages.

We also define the following probability.

***Definition 1*** (*Leaving probability*) Let $u$ be an arbitrary user of the group. Assume that the rekey interval is fixed. Then $u$'s *leaving probability* $p(u)$ is the probability that $u$ leaves the group in the rekey interval.

The leaving probability is an average number of leaving the group in a rekey interval. The probability is calculated as follows:

$$p(u) = \frac{(\text{The number of } u\text{'s leaving the group})}{\left(\begin{array}{c}\text{The number of rekey intervals}\\ \text{in which } u \text{ subscribes}\end{array}\right)}$$

For the new subscriber, however, because the numerator and the denominator are $0$, we cannot derive its leaving probability. In this case, we set the probability as the average leaving probability of the entire users. We will introduce the way for determining the leaving probability in Section 5. If a user has a higher leaving probability, the user leaves and joins more frequently. This leaving probability determines the structure of the optimal key-tree.

## 3 Analysis of Communication Cost for Group Rekeying

The communication cost owing to group rekeying depends on the size of the rekeying messages. Because the cases for users' leaving are diverse, we calculate the average size of rekeying messages.

***Definition 2*** (*Average size of rekeying messages*) For a key-tree $\mathbf{T}$, $\mathcal{M}(\mathbf{T})$ is the average size of rekeying messages.

Let $\mathcal{U}_{\mathbf{T}} = \{u_1, u_2, \cdots, u_N\}$, where $N$ is the group size. Also let $q(u_i) = 1 - p(u_i)$; then $q(u_i)$ is the probability that $u_i$ remains in the group in a rekey interval. For $k_a \in \mathcal{K}_{\mathbf{T}}$ and $k_b \in \mathcal{K}_{\mathbf{T}}$, $k_b \prec k_a$ denotes that $k_a$ is the parent of $k_b$. When $k_b \prec k_a$, let $\{k_a\}_{k_b}$ be $k_a$ encrypted by $k_b$. Let $\mathcal{K}_{\mathbf{T}}^e$ be the set of all encrypted keys. Then,

$$\mathcal{K}_{\mathbf{T}}^e \triangleq \left\{\{k_a\}_{k_b} | k_a,\ k_b \in \mathcal{K}_{\mathbf{T}},\ k_b \prec k_a\right\}.$$

The rekeying message consists of some elements of $\mathcal{K}_{\mathbf{T}}^e$.

Let $\mathcal{C}$ be the set of possible cases of users' leaving. Then $|\mathcal{C}| = 2^N$ because there are $N$ users and each user has two choices: leave or remain. Consider the mapping $V : \mathcal{K}_{\mathbf{T}}^e \times \mathcal{C} \to \{0, 1\}$. $V(\{k_a\}_{k_b}, c)$ is 1 if and only if there exists $\{k_a\}_{k_b}$ in the rekeying message when case $c$ has occurred. Let $L(c)$ be the size of rekeying messages for case $c$. Then,

$$L(c) = \sum_{\{k_a\}_{k_b} \in \mathcal{K}_{\mathbf{T}}^e} V(\{k_a\}_{k_b}, c).$$

In Fig 4 (b), the total length in the rightmost column indicates $L(c)$. Let $Pr(c)$ be the probability of case $c$ occurring. Because $\mathcal{M}(\mathbf{T})$ is the average value of $L(c)$, $\mathcal{M}(\mathbf{T})$ is calculated as follows.

$$\mathcal{M}(\mathbf{T}) = \sum_{c \in \mathcal{C}} [Pr(c) \cdot L(c)] = \sum_{\{k_a\}_{k_b} \in \mathcal{K}_{\mathbf{T}}^e} \left[\sum_{V(\{k_a\}_{k_b}, c) = 1} Pr(c)\right] \quad (1)$$

The term $\sum_{V(\{k_a\}_{k_b}, c)=1} Pr(c)$ represents the sum of the probabilities of the occurrence of $c$, which contains encrypted key $\{k_a\}_{k_b}$. For the encrypted key $\{k_a\}_{k_b} \in \mathcal{K}_e$, let $\Pr(\{k_a\}_{k_b})$ be the probability that $\{k_a\}_{k_b}$ exists in the rekeying message. Then $\Pr(\{k_a\}_{k_b}) = \sum_{V(\{k_a\}_{k_b}, c)=1} Pr(c)$. In Fig 4-(b), the average number of keys in the lowest row indicates $\Pr(\{k_a\}_{k_b})$.

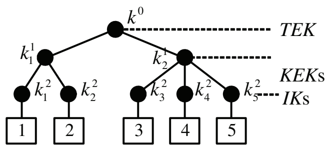
$$\therefore \mathcal{M}(\mathbf{T}) = \sum_{\{k_a\}_{k_b} \in \mathcal{K}_{\mathbf{T}}^e} \Pr(\{k_a\}_{k_b}). \quad (2)$$

Now, let us calculate $\Pr(\{k_a\}_{k_b})$. The necessary and sufficient condition that $\{k_a\}_{k_b}$ does not occur in a rekeying messages is that one of following two conditions is satisfied:

a) All users who have key $k_a$ remain
b) All users who have key $k_b$ leave.

The probability that condition a) is satisfied is $\prod_{u \in \mathcal{U}_{\mathbf{T}}(k_a)} q(u)$, and the probability that condition b) is satisfied is $\prod_{u \in \mathcal{U}_{\mathbf{T}}(k_b)} p(u)$. Because these two cases are

www.redpel.com
+917620593389



(a)

| Cases ($c$) | The number of keys | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\{k^0\}_{k_1^1}$ | $\{k^0\}_{k_2^1}$ | $\{k_1^1\}_{k_1^2}$ | $\{k_1^1\}_{k_2^2}$ | $\{k_2^1\}_{k_3^2}$ | $\{k_2^1\}_{k_4^2}$ | $\{k_2^1\}_{k_5^2}$ | $L(c)$ |
| Only user 1 leaves | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 3 |
| Only user 2 leaves | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 3 |
| Only user 3 leaves | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 4 |
| …… | | | | | | | | |
| All users leave | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Average ($\Sigma$Pr $k$) | $\Pr\left(\{k^0\}_{k_1^1}\right)$ | $\Pr\left(\{k^0\}_{k_2^1}\right)$ | $\Pr\left(\{k_1^1\}_{k_1^2}\right)$ | $\Pr\left(\{k_1^1\}_{k_2^2}\right)$ | $\Pr\left(\{k_2^1\}_{k_3^2}\right)$ | $\Pr\left(\{k_2^1\}_{k_4^2}\right)$ | $\Pr\left(\{k_2^1\}_{k_5^2}\right)$ | $M(\mathbf{T})$ |

(b)

Fig. 4. (a) The example of the *LKH* for $5$ users
(b) The calculation method of average size of rekeying messages

mutually exclusive,

$$\therefore \mathcal{M}(\mathbf{T}) = \sum_{\{k_a\}_{k_b} \in \mathcal{K}_{\mathbf{T}}^e} \left[1 - Q_{\mathbf{T}}(k_a) - P_{\mathbf{T}}(k_b)\right], \qquad (3)$$

where $P_{\mathbf{T}}(k) \triangleq \prod_{u \in \mathcal{U}_{\mathbf{T}}(k)} p(u)$, and $Q_{\mathbf{T}}(k) \triangleq \prod_{u \in \mathcal{U}_{\mathbf{T}}(k)} q(u)$.

Let $\mathcal{K}_{\mathbf{T}}^T$, $\mathcal{K}_{\mathbf{T}}^K$, and $\mathcal{K}_{\mathbf{T}}^I$ be the set of the entire TEK, all KEKs, and all IKs, respectively. And let $c_{\mathbf{T}}(k)$ be the number of $k$'s child nodes in the key-tree $\mathbf{T}$. Because $k_a \in \mathcal{K}_{\mathbf{T}}^T \cup \mathcal{K}_{\mathbf{T}}^K$ and $k_b \in \mathcal{K}_{\mathbf{T}}^K \cup \mathcal{K}_{\mathbf{T}}^I$ in $\{k_a\}_{k_b}$,

$$\mathcal{M}(\mathbf{T}) = \sum_{k \in \mathcal{K}_{\mathbf{T}}^K \cup \mathcal{K}_{\mathbf{T}}^I} \left[1 - P_{\mathbf{T}}(k)\right] - \sum_{k \in \mathcal{K}_{\mathbf{T}}^T \cup \mathcal{K}_{\mathbf{T}}^K} \left[c_{\mathbf{T}}(k) Q_{\mathbf{T}}(k)\right] \qquad (4)$$

For IKs and TEK,

$$\sum_{k \in \mathcal{K}_{\mathbf{T}}^I} \left[1 - P_{\mathbf{T}}(k)\right] = N - \sum_{u \in \mathcal{U}_{\mathbf{T}}} p(u) = \sum_{u \in \mathcal{U}_{\mathbf{T}}} q(u), \qquad (5)$$

$$\sum_{k \in \mathcal{K}_{\mathbf{T}}^T} \left[c_{\mathbf{T}}(k) \prod_{u \in \mathcal{U}_{\mathbf{T}}(k)} q(u)\right] = c_{\mathbf{T}}(\text{TEK}) \prod_{u \in \mathcal{U}_{\mathbf{T}}} q(u). \qquad (6)$$

Also, for a key-node $k$, let us define $R_{\mathbf{T}}(k)$ as follows:

$$R_{\mathbf{T}}(k) \triangleq P_{\mathbf{T}}(k) + c_{\mathbf{T}}(k) \cdot Q_{\mathbf{T}}(k).$$

From equation (5) and (6), we finally derive $\mathcal{M}(\mathbf{T})$ as follows:

$$\mathcal{M}(\mathbf{T}) = \sum_{u \in \mathcal{U}_{\mathbf{T}}} q(u) - c_{\mathbf{T}}(\text{TEK}) \prod_{u \in \mathcal{U}_{\mathbf{T}}} q(u) + \sum_{k \in \mathcal{K}_{\mathbf{T}}^K} \left[1 - R_{\mathbf{T}}(k)\right]. \qquad (7)$$

Equation (7) consists of the sum of $R_{\mathbf{T}}(k)$, where $k$ is a KEK key-node. For each KEK key-node $k$, $R_{\mathbf{T}}(k)$ consists of the product of leaving probabilities $[P_{\mathbf{T}}(k)]$, the product of remaining probabilities$[Q_{\mathbf{T}}(k)]$, and the number of child key-nodes$[c_{\mathbf{T}}(k)]$. Thus, $\mathcal{M}(\mathbf{T})$ depends on both the shape of the tree structure and the locations of the user-nodes.

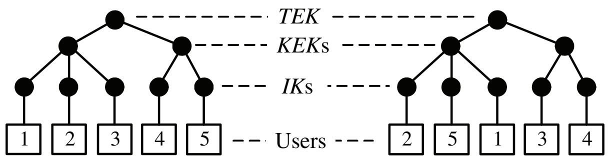


Fig. 5. An example of the similar trees.

***Example.*** Let us calculate directly the average size of the rekeying messages for the key tree in Fig. 4-(a) by enumerating all the possible leaving cases. Assume that the leaving probabilities of user 1, 2, 3, 4, and 5(who are mapped to $u_1$, $u_2$, $u_3$, $u_4$, and $u_5$, respectively) are 0.1, 0.2, 0.2, 0.25, and 0.3, respectively. Then, $q(u_1) = 0.9$, $q(u_2) = q(u_3) = 0.8$, $q(u_4) = 0.75$, and $q(u_5) = 0.7$. Because there are five users, there are $2^5 = 32$ leaving scenarios. According to the table in Fig. 4-(b), the average size of the rekeying messages for the 32 leaving cases is calculated as follows:

$$\sum_{c \in \mathcal{C}} Pr(c) \cdot L(c) = 0.0336 \times 3 + 0.0756 \times 3 + \cdots = 2.6102.$$

We can also compute the average size of the rekeying messages by using equation (7) as follows:

$$\sum_{u \in \mathcal{U}_{\mathbf{T}}} q(u) = 3.95, \quad \prod_{u \in \mathcal{U}_{\mathbf{T}}} q(u) = 0.3024,$$

$$R_{\mathbf{T}}(k_1^1) = 1.46, \quad R_{\mathbf{T}}(k_2^1) = 1.275.$$

$$\therefore \mathcal{M}(\mathbf{T}) = 3.95 - 2 \times 0.3024 + (1 - 1.46) + (1 - 1.275) = 2.6102.$$

The comparison demonstrates that the result obtained from equation (7) is the same as the result of the direct average calculation.

## 4 Optimal Key-Tree

In this section, we discuss the conditions for the efficient key-tree. For this tree $\mathbf{T}$, the average size of the rekeying messages, $\mathcal{M}(\mathbf{T})$, should be minimized. Following the analysis in the previous section, we now investigate the locations of the users and the shape of the key-tree structure.

First, we define the efficiency of the key-tree as follows.

***Definition 3*** *(Efficiency)* Let $\mathbf{T}$ and $\mathbf{T}'$ be two key-trees, where $\mathcal{U}_{\mathbf{T}} = \mathcal{U}_{\mathbf{T}'}$. $\mathbf{T}$ is more *efficient* than $\mathbf{T}'$ if and only if $\mathcal{M}(\mathbf{T}) < \mathcal{M}(\mathbf{T}')$.

### 4.1 Locations of User-Nodes on a Tree Structure

In this subsection, we discuss the locations of users on the tree structure which is one of the key components determining the average size of rekeying messages. We find some conditions for locating users on the tree structure for the efficient key-tree. Let $\mathcal{K}_{\mathbf{T}}(k)$ be the set of key-nodes which are the children of $k$.

***Definition 4*** *(Similar structures)* Let $\mathbf{T}$ and $\mathbf{T}'$ be two

www.redpel.com
+917620593389

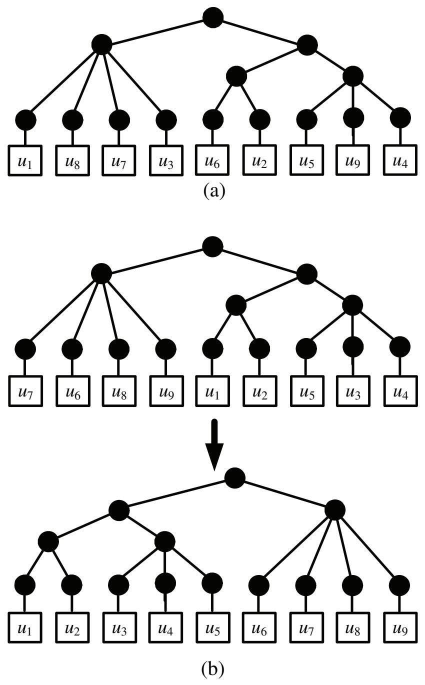

$u_1$ $u_8$ $u_7$ $u_3$ $u_6$ $u_2$ $u_5$ $u_9$ $u_4$
(a)
$u_7$ $u_6$ $u_8$ $u_9$ $u_1$ $u_2$ $u_5$ $u_3$ $u_4$
$u_1$ $u_2$ $u_3$ $u_4$ $u_5$ $u_6$ $u_7$ $u_8$ $u_9$
(b)

Fig. 6. Examples of the non-ordered (a) and ordered trees (b).($p(u_1) \leq p(u_2) \leq \cdots \leq p(u_9)$)

key-trees. $\mathbf{T}$ and $\mathbf{T}'$ are *similar structures* if and only if following properties are established.

a) $\mathcal{U}_{\mathbf{T}} = \mathcal{U}_{\mathbf{T}'}$, $\mathcal{K}_{\mathbf{T}} = \mathcal{K}_{\mathbf{T}'}$
b) $\forall k \in \mathcal{K}_{\mathbf{T}} = \mathcal{K}_{\mathbf{T}'}$ except when $k$ is an IK key-node, $\mathcal{K}_{\mathbf{T}}(k) = \mathcal{K}_{\mathbf{T}'}(k)$.

Fig. 5 shows an example of similar structures. Two similar key-trees have the same tree structure, and only the orders of user-nodes differ. Now, to investigate the properties of the efficient key-tree, let us define the following notations.

***Definition 5*** *(Ordered node)* Let $\mathbf{T}$ be a key-tree and $k$ be a key-node of $\mathbf{T}$. Node $k$ is an *ordered node* if and only if for any two child nodes $k_a$ and $k_b$ of $k$, one of the following two propositions is satisfied:

$$\forall u_i \in \mathcal{U}_{\mathbf{T}}(k_a) \text{ and } u_j \in \mathcal{U}_{\mathbf{T}}(k_b),\ p(u_i) \leq p(u_j), \text{ or}$$
$$\forall u_i \in \mathcal{U}_{\mathbf{T}}(k_a) \text{ and } u_j \in \mathcal{U}_{\mathbf{T}}(k_b),\ p(u_i) \geq p(u_j)$$

***Definition 6*** *(Ordered tree)* Let $\mathbf{T}$ be a key-tree. $\mathbf{T}$ is an *ordered tree* if and only if $\forall k \in \mathcal{K}_{\mathbf{T}}$, and $k$ is an ordered node.

If $\mathbf{T}$ is an ordered tree, we can rearrange its user-nodes by increasing order of users' leaving probabilities without changing the structure of $\mathbf{T}$. A tree in Fig.6 (a) is not an ordered tree. But a tree in (b) is an ordered tree because it satisfies Definition 6. This is also because the upper tree in Fig.6 (b) can be rearranged to the lower tree, whose user-nodes are arranged in increasing order of users' leaving probabilities.

***Lemma 1*** Let $k_a$ and $k_b$ be two KEK key-nodes of key-tree $\mathbf{T}$ that are not in the same key path. Assume the following statements:

a) $\mathcal{U}_{\mathbf{T}}(k_a) \cup \mathcal{U}_{\mathbf{T}}(k_b) = \{u_1, u_2, \cdots, u_n\}$, where $n$ is a fixed positive integer, and $n \geq 2$.
b) $|\mathcal{U}_{\mathbf{T}}(k_a)| = m$, $|\mathcal{U}_{\mathbf{T}}(k_b)| = n - m$, where $m$ is a fixed positive integer, and $1 \leq m \leq n-1$.

Then, $R_{\mathbf{T}}(k_a) + R_{\mathbf{T}}(k_b)$ is maximized if and only if

$$\forall u_i \in \mathcal{U}_{\mathbf{T}}(k_a) \text{ and } u_j \in \mathcal{U}_{\mathbf{T}}(k_b),\ p(u_i) \geq p(u_j),$$
$$\text{or } \forall u_i \in \mathcal{U}_{\mathbf{T}}(k_a) \text{ and } u_j \in \mathcal{U}_{\mathbf{T}}(k_b),\ p(u_i) \leq p(u_j).$$

**pf.** Because $p(u) \ll q(u)$ in most cases, $P_{\mathbf{T}}(k_a) \ll Q_{\mathbf{T}}(k_a)$ and $P_{\mathbf{T}}(k_b) \ll Q_{\mathbf{T}}(k_b)$, where $u$ is an arbitrary user. Hence, we may consider that $c_{\mathbf{T}}(k_a)Q_{\mathbf{T}}(k_a) + c_{\mathbf{T}}(k_b)Q_{\mathbf{T}}(k_b)$ maximizes $R_{\mathbf{T}}(k_a) + R_{\mathbf{T}}(k_b)$.

Consider the case in which $c_{\mathbf{T}}(k_a)Q_{\mathbf{T}}(k_a) + c_{\mathbf{T}}(k_b)Q_{\mathbf{T}}(k_b)$ is maximized. Without loss of generality, let $c_{\mathbf{T}}(k_a)Q_{\mathbf{T}}(k_a) \geq c_{\mathbf{T}}(k_b)Q_{\mathbf{T}}(k_b)$. Assume that there exist two users $u_a \in \mathcal{U}_{\mathbf{T}}(k_a)$ and $u_b \in \mathcal{U}_{\mathbf{T}}(k_b)$ such that $p(u_a) > p(u_b)$. Then $q(u_a) < q(u_b)$ and $\frac{c_{\mathbf{T}}(k_a)Q_{\mathbf{T}}(k_a)}{q(u_a)} > \frac{c_{\mathbf{T}}(k_b)Q_{\mathbf{T}}(k_b)}{q(u_b)}$.

$$\left[Q_{\mathbf{T}}(k_a) \cdot \frac{q(u_b)}{q(u_a)} + Q_{\mathbf{T}}(k_b) \cdot \frac{q(u_a)}{q(u_b)}\right] - [Q_{\mathbf{T}}(k_a) + Q_{\mathbf{T}}(k_b)]$$
$$= \left[\frac{Q_{\mathbf{T}}(k_a)}{q(u_a)} - \frac{Q_{\mathbf{T}}(k_b)}{q(u_b)}\right][q(u_b) - q(u_a)] > 0$$

The above inequality indicates that when $u_b$ and $u_a$ are switched, i.e., $u_b \in \mathcal{U}_{\mathbf{T}}(k_a)$ and $u_a \in \mathcal{U}_{\mathbf{T}}(k_b)$, $c_{\mathbf{T}}(k_a)Q_{\mathbf{T}}(k_a) + c_{\mathbf{T}}(k_b)Q_{\mathbf{T}}(k_b)$ is larger than the original case. This is contradiction to the property of the maximum for $c_{\mathbf{T}}(k_a)Q_{\mathbf{T}}(k_a) + c_{\mathbf{T}}(k_b)Q_{\mathbf{T}}(k_b)$. Hence, $\forall u_a \in \mathcal{U}_{\mathbf{T}}(k_a)$ and $u_b \in \mathcal{U}_{\mathbf{T}}(k_b)$, $p(u_a) \leq p(u_b)$. ■

***Lemma 2*** Let $\mathbf{T}$ and $k$ be a key-tree and its key-node for a KEK or TEK. To maximize $\sum_{k_i \in \mathcal{K}_{\mathbf{T}}(k)} R_{\mathbf{T}}(k_i)$, key-node $k$ should be well ordered.

**pf.** Assume that there exists a case in which $\sum_{k_i \in \mathcal{K}_{\mathbf{T}}(k)} R_{\mathbf{T}}(k_i)$ is maximized even though $k$ is not well ordered. Then, there are two key-nodes $k_\alpha$, $k_\beta \in \mathcal{K}_{\mathbf{T}}(k)$ that satisfy the existence of $u_a$, $u_b \in \mathcal{U}_{\mathbf{T}}(k_\alpha)$ and $u_c$, $u_d \in \mathcal{U}_{\mathbf{T}}(k_\beta)$, such that $p(u_a) > p(u_c)$ and $p(u_b) < p(u_d)$.

However, according to *Lemma 1*, $R_{\mathbf{T}}(k_\alpha) + R_{\mathbf{T}}(k_\beta)$ is maximized when $\forall u_e \in \mathcal{U}_{\mathbf{T}}(k_\alpha)$ and $u_f \in \mathcal{U}_{\mathbf{T}}(k_\beta)$, $p(u_e) \geq p(u_f)$ or $\forall u_e \in \mathcal{U}_{\mathbf{T}}(k_\alpha)$ and $u_f \in \mathcal{U}_{\mathbf{T}}(k_\beta)$, $p(u_e) \leq p(u_f)$. This is contradiction to the property that $\sum_{k_i \in \mathcal{K}_{\mathbf{T}}(k)} R_{\mathbf{T}}(k_i)$ is a maximum. ■

***Theorem 1*** Let $\mathbf{T}$ be a key-tree. Then, $\mathbf{T}$ is the most efficient tree among trees similar to $\mathbf{T}$ if and only if $\mathbf{T}$ is ordered.

**pf.** Assume that there exists a case in which $\mathbf{T}'$ is the most efficient tree among trees similar to $\mathbf{T}'$, but $\mathbf{T}'$ is not ordered.

From the definition, there is a $k$ in $\mathbf{T}'$ that is not ordered. From *Lemma 2*, $f(k)$ increases when $k$ becomes ordered. Hence, when we modify tree $\mathbf{T}'$ as $k$ becomes

www.redpel.com
+917620593389

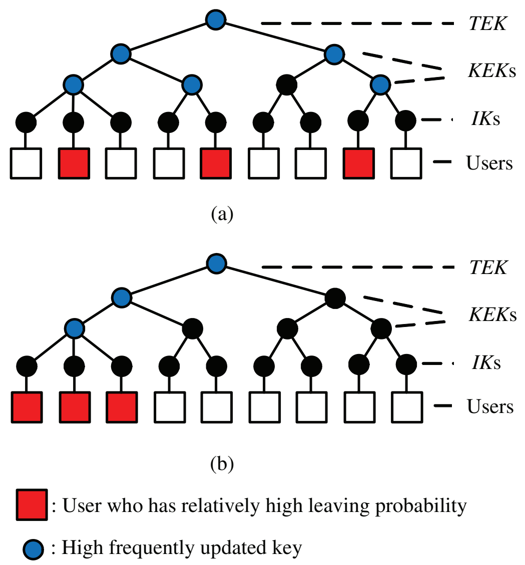

TEK
KEKs
IKs
Users
(a)
TEK
KEKs
IKs
Users
(b)
🟥 : User who has relatively high leaving probability
● : High frequently updated key

Fig. 7. Example of two cases: (a) not-ordered and (b) ordered trees.

ordered, we get a more efficient key-tree. This is contradiction to the property that $\mathbf{T}'$ is most efficient. Therefore, if $\mathbf{T}$ is the most efficient tree among trees similar to $\mathbf{T}$, $\mathbf{T}$ is ordered. ■

According to Theorem 1, a KEK key-node in a key-tree should be shared only by users whose leaving probabilities are similar for the efficient key-tree. Actually, if a user has relatively high leaving probabilities, the possibility that the KEK which is shared by the user is updated becomes higher. If the relatively high-probability users are deployed widely in the tree structure, the number of frequently updated KEKs increases. On the other hand, if the relatively high-probability users are concentrated in the tree structure, the number of frequently updated KEKs decreases. Fig.7 compares an unordered and an ordered tree. As the figure shows, if the users who are likely to leave in a rekey interval are dispersed in the tree structure, the number of updated keys increases on average.

## 4.2 Shape of Key-Tree Structure

In this subsection, we discuss the shape of the key-tree structure, which is another element that determines the average size of rekeying messages. We propose conditions for the shape of the tree structure for the efficient key-tree. We assume that such a tree should be ordered, as proven in the previous subsection.

The proposed conditions consist of four standards:

1) Splitting a KEK-node into two KEK key-nodes,
2) Merging two sibling KEK key-nodes,
3) Creating new child key-nodes, and
4) Deallocating the child key-nodes.

***Definition 7*** *(Neighbor)* Let $\mathbf{T}$ be an ordered key-tree and $k_a$ and $k_b$ be two sibling key-nodes. $k_a$ and $k_b$ are *neighbors* if and only if for any $u_i \in \mathcal{U}_\mathbf{T}(k_a)$ and $u_j \in \mathcal{U}_\mathbf{T}(k_b)$(without loss of generality, assume that $p(u_i) \leq p(u_j)$), there exists no user $u_l$ such that $u_l \notin \mathcal{U}_\mathbf{T}(k_a) \cup \mathcal{U}_\mathbf{T}(k_b)$ and $p(u_i) < p(u_l) < p(u_j)$.

The logical meaning of neighbor is as follows. For an ordered tree, we can sort the order of users by the probabilities that they leave in a rekey interval. In the sorted tree, $k_a$ and $k_b$ are neighbors if and only if $k_a$ and $k_b$ are sibling key-nodes and one is right next to the other.

***Theorem 2*** *(Splitting KEK)* Let $\mathbf{T}$ and $k$ be a key-tree and a KEK, respectively, and $k_p$ be a parent key-node of $k$. Also, let $\mathcal{K}_\mathbf{T}(k) = \{k_{e_1}, k_{e_2}, \cdots, k_{e_{c_\mathrm{T}(k)}}\}$, $c_\mathrm{T}(k) \geq 4$, and $k_{e_1}$, $k_{e_2}$, $\cdots$, $k_{e_{c_\mathrm{T}(k)}}$ be ordered key-nodes in this order, i.e.,

$$\forall u_1 \in \mathcal{U}_\mathbf{T}(k_{e_1}),\, u_2 \in \mathcal{U}_\mathbf{T}(k_{e_2}),\, \cdots,\, u_{c_\mathrm{T}(k)} \in \mathcal{U}_\mathbf{T}(k_{e_{c_\mathrm{T}(k)}}),$$
$$p(u_1) \leq p(u_2) \leq \cdots \leq p(u_{c_\mathrm{T}(k)}).$$

Consider the case in which $\mathbf{T}$ is modified to $\mathbf{T}'(d)$ as

$$\mathcal{K}_{\mathbf{T}'(d)}(k_p) = [\mathcal{K}_\mathbf{T}(k_p) \backslash \{k\}] \cup \{k_a^d, k_b^d\},$$
$$\mathcal{K}_{\mathbf{T}'(d)}(k_a^d) = \{k_{e_1}, k_{e_2}, \cdots, k_{e_d}\}, \text{ and}$$
$$\mathcal{K}_{\mathbf{T}'(d)}(k_b^d) = \{k_{e_{d+1}}, k_{e_{d+2}}, \cdots, k_{e_{c_\mathrm{T}(k)}}\},$$

where $d$ is a positive integer and $1 \leq d < c_\mathrm{T}(k)$. Let $\hat{d} = \arg\max_d [R_{\mathbf{T}'(d)}(k_a^d) + R_{\mathbf{T}'(d)}(k_b^d)]$. If

$$R_{\mathbf{T}'(\hat{d})}(k_a^{\hat{d}}) + R_{\mathbf{T}'(\hat{d})}(k_b^{\hat{d}}) + Q_\mathbf{T}(k_p) > R_\mathbf{T}(k) + 1, \tag{8}$$

then $\mathbf{T}'(\hat{d})$ is a more efficient tree than $\mathbf{T}$.

**pf.** From Lemma 1, the values of $R_{\mathbf{T}'(d)}(k_a^d) + R_{\mathbf{T}'(d)}(k_b^d)$ are candidates for maximum values of $R_\mathbf{T}(k_a) + R_\mathbf{T}(k_b)$, where $k_a$ and $k_b$ are split keys from $k$, because $\forall u_i \in \mathcal{U}_{\mathbf{T}'(d)}(k_a^d)$ and $u_j \in \mathcal{U}_{\mathbf{T}'(d)}(k_b^d)$, $p_i \leq p_j$. Then, $R_{\mathbf{T}'(\hat{d})}(k_a^{\hat{d}}) + R_{\mathbf{T}'(\hat{d})}(k_b^{\hat{d}})$ is maximum value of $R_\mathbf{T}(k_a) + R_\mathbf{T}(k_b)$.

When $k$ is split into $k_a$ and $k_b$, the changed terms of equation (7) are $1 - R_\mathbf{T}(k)$ and $1 - R_\mathbf{T}(k_p)$. $1 - R_\mathbf{T}(k)$ is modified into $[1 - R_{\mathbf{T}'(\hat{d})}(k_a)] + [1 - R_{\mathbf{T}'(\hat{d})}(k_b)]$, and $1 - R_\mathbf{T}(k_p)$ is into $1 - R_{\mathbf{T}'(\hat{d})}(k_p)$. If

$$[1 - R_{\mathbf{T}'(\hat{d})}(k_a)] + [1 - R_{\mathbf{T}'(\hat{d})}(k_b)] + [1 - R_{\mathbf{T}'(\hat{d})}(k_p)] < [1 - R_\mathbf{T}(k)] + [1 - R_\mathbf{T}(k_p)], \tag{9}$$

$\mathbf{T}'(\hat{d})$ is more efficient than $\mathbf{T}$. Also, because $P_{\mathbf{T}'(\hat{d})}(k_p) = P_\mathbf{T}(k_p)$, $Q_{\mathbf{T}'(\hat{d})}(k_p) = Q_\mathbf{T}(k_p)$, and $c_{\mathbf{T}'(\hat{d})}(k_p) = c_\mathbf{T}(k_p) + 1$, inequality (9) is summarized as inequality (8). ■

The above theorem is a standard that can determine whether the KEK should be split for the efficiency. For an arbitrary key $k$ except the IK, if the number of the users who share $k$ increases, the number of the child key-nodes of $k$ may also increase. Because the number of encrypted new $k$ in the rekeying message is the same as the number of child key-nodes of $k$ when $k$ should be updated, the case that $k$ is split is more efficient. Of course, $k$ is split while the key-tree maintains the ordered tree.

***Corollary 3*** *(Merging two neighbor KEKs)* Let $\mathbf{T}$ be a key-tree, $k_a$ and $k_b$ be neighbor key-nodes, and $k_p$ be a parent key-node of $k_a$ and $k_b$. Let $c_\mathrm{T}(k_p) \geq 3$. Consider the case

www.redpel.com
+917620593389

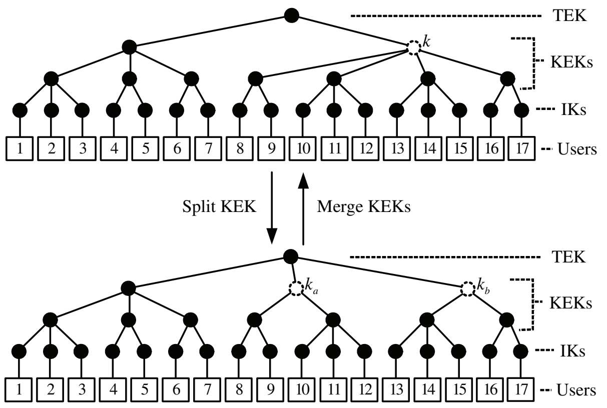


Fig. 8. Example of splitting and merging

in which $\mathbf{T}$ is modified to $\mathbf{T}'$ as $k_a$ and $k_b$ are merged into $k$, i.e.,

$$\begin{aligned}\mathcal{K}_{\mathbf{T}'}(k_p) &= [\mathcal{K}_{\mathbf{T}}(k_p)\backslash\{k_a,k_b\}]\cup\{k\},\\ \mathcal{K}_{\mathbf{T}'}(k) &= \mathcal{K}_{\mathbf{T}}(k_a)\cup\mathcal{K}_{\mathbf{T}}(k_b).\end{aligned}$$

If $R_{\mathbf{T}}(k_a)+R_{\mathbf{T}}(k_b)+Q_{\mathbf{T}}(k_p) < R_{\mathbf{T}}(k)+1$, then $\mathbf{T}'$ is a more efficient tree than $\mathbf{T}$.

This corollary is the reverse case of theorem 2. Hence, we may skip the proof. If the numbers of users who share $k_a$ or $k_b$ decreases, the difference in the average number of encrypted $k_p$ may become larger than the difference between the average size of encrypted $k$ and the sum of the average size of encrypted $k_a$ and $k_b$ as $\mathbf{T}$ is modified to $\mathbf{T}'$. Also, because the merged key-nodes are neighbors, the key-tree is still an ordered tree.

***Theorem 4*** (Creating child key-nodes) Let $\mathbf{T}$ and $k$ be a key-tree and a KEK or TEK key-node, respectively. Also, let $c_{\mathbf{T}}(k)\geq 4$, $\mathcal{K}_{\mathbf{T}}(k)=\{k_{e_1},k_{e_2},\cdots,k_{e_{c_{\mathbf{T}}(k)}}\}$, and $k$ be an ordered key-node, i.e.,

$$\forall u_1\in\mathcal{U}_{\mathbf{T}}(k_{e_1}),\ u_2\in\mathcal{U}_{\mathbf{T}}(k_{e_2}),\ \cdots,\ u_{c_{\mathbf{T}}(k)}\in\mathcal{U}_{\mathbf{T}}(k_{e_{c_{\mathbf{T}}(k)}}),\\ p(u_1)\leq p(u_2)\leq\cdots\leq p(u_{c_{\mathbf{T}}(k)}).$$

Consider the case in which $\mathbf{T}$ is modified to $\mathbf{T}'(d)$ as

$$\begin{aligned}\mathcal{K}_{\mathbf{T}'(d)}(k) &= \{k_a^d,k_b^d\},\\ \mathcal{K}_{\mathbf{T}'(d)}(k_a^d) &= \{k_{e_1},k_{e_2},\cdots,k_{e_d}\},\\ \mathcal{K}_{\mathbf{T}'(d)}(k_b^d) &= \{k_{e_{d+1}},k_{e_{d+2}},\cdots,k_{e_{c_{\mathbf{T}}(k)}}\},\end{aligned}$$

where $d$ is a positive integer and $2\leq d\leq c_{\mathbf{T}}(k)-2$. Let $\hat{d}=\arg\max_d[R_{\mathbf{T}'(d)}(k_a^d)+R_{\mathbf{T}'(d)}(k_b^d)]$. If

$$R_{\mathbf{T}'(\hat{d})}(k_a^{\hat{d}})+R_{\mathbf{T}'(\hat{d})}(k_b^{\hat{d}}) > 2+[c_{\mathbf{T}}(k)-2]\cdot Q_{\mathbf{T}}(k), \quad (10)$$

then $\mathbf{T}'(\hat{d})$ is a more efficient tree than $\mathbf{T}$.

**pf.** From Lemma 1, the values of $R_{\mathbf{T}'(d)}(k_a^d)+R_{\mathbf{T}'(d)}(k_b^d)$ are candidates for the maximum value among all possible $R_{\mathbf{T}'}(k_a)+R_{\mathbf{T}'}(k_b)$ values, where $\mathcal{K}_{\mathbf{T}'}(k)=\{k_a,k_b\}$, $\mathcal{K}_{\mathbf{T}}(k)=\mathcal{K}_{\mathbf{T}'}(k_a)\cup\mathcal{K}_{\mathbf{T}'}(k_b)$, and $\mathcal{K}_{\mathbf{T}'}(k_a)\cap\mathcal{K}_{\mathbf{T}'}(k_b)=\emptyset$. Hence, $R_{\mathbf{T}'(\hat{d})}(k_a^{\hat{d}})+R_{\mathbf{T}'(\hat{d})}(k_b^{\hat{d}})$ is the maximum value of $R_{\mathbf{T}'}(k_a)+R_{\mathbf{T}'}(k_b)$.

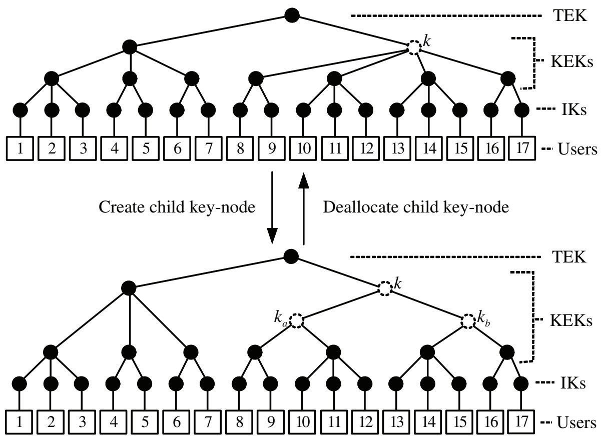


Fig. 9. Example of creating and deallocating child key-nodes

The terms of equation (7) that differ between $\mathcal{M}(\mathbf{T})$ and $\mathcal{M}(\mathbf{T}')$ are those for $k$, $k_a$, and $k_b$. The term for $k$ is modified from $1-R_{\mathbf{T}}(k)$ to $1-P_{\mathbf{T}'(\hat{d})}(k)-c_{\mathbf{T}'(\hat{d})}(k)Q_{\mathbf{T}'(\hat{d})}(k)$. Also, for $k_a^{\hat{d}}$ and $k_b^{\hat{d}}$, two terms, $[1-R_{\mathbf{T}'(\hat{d})}(k_a^{\hat{d}})]$ and $[1-R_{\mathbf{T}'(\hat{d})}(k_b^{\hat{d}})]$ are newly created. If

$$\begin{aligned}1-R_{\mathbf{T}}(k) > [1-P_{\mathbf{T}'(\hat{d})}(k)-c_{\mathbf{T}'(\hat{d})}(k)Q_{\mathbf{T}'(\hat{d})}(k)]\\ +[1-R_{\mathbf{T}'(\hat{d})}(k_a^{\hat{d}})]+[1-R_{\mathbf{T}'(\hat{d})}(k_b^{\hat{d}})],\end{aligned} \quad (11)$$

$\mathbf{T}'(d)$ is more efficient than $\mathbf{T}$. Also, because $P_{\mathbf{T}'(\hat{d})}(k_p)=P_{\mathbf{T}}(k_p)$, $Q_{\mathbf{T}'(\hat{d})}(k_p)=Q_{\mathbf{T}}(k_p)$, and $c_{\mathbf{T}'(\hat{d})}(k)=2$, inequality (11) is summarized as inequality (10). ■

The above theorem is a standard that can determine whether we create child key-nodes of $k$ for efficiency, where $k$ is a KEK or TEK. If the number of the users who share $k$ increases, the number of child key-nodes of $k$ may also increase. As in key-splitting, this theorem can be applied when the number of users who share $k$ increases. Hence, we may choose whichever of Theorem 2 or Theorem 4 which reduces communication overhead more when $\mathcal{U}_{\mathbf{T}}(k)$ becomes large.

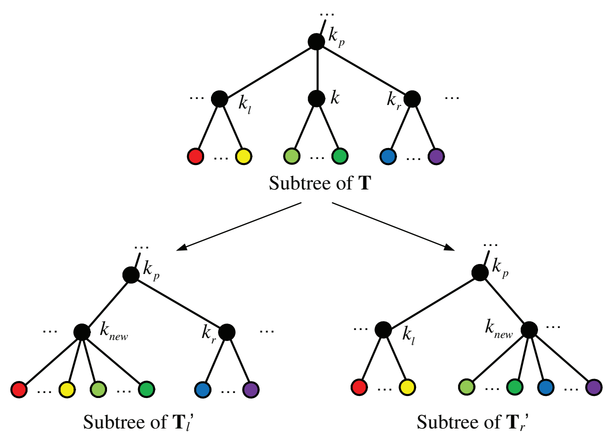
***Corollary 5*** (Deallocating child key-nodes) Let $\mathbf{T}$ be a key-tree and $k$ be a KEK key-node in $\mathbf{T}$, where $\mathcal{K}_{\mathbf{T}}(k)=\{k_a,k_b\}$. Consider the case in which $\mathbf{T}$ is modified to $\mathbf{T}'$, such that $\mathcal{K}_{\mathbf{T}'}(k)=\mathcal{K}_{\mathbf{T}}(k_a)\cup\mathcal{K}_{\mathbf{T}}(k_b)$ and $k_a$ and $k_b$ are eliminated. If

$$R_{\mathbf{T}}(k_a)+R_{\mathbf{T}}(k_b) < 2+[c_{\mathbf{T}}(k)-2]\cdot Q_{\mathbf{T}}(k),$$

then $\mathbf{T}'$ is more efficient than $\mathbf{T}$.

This corollary is the reverse case of Theorem 4. Hence, we may skip the proof. Like Corollary 3, this corollary can be applied when the number of users who shares $k$ decreases, where $k$ is a key-node for a TEK or KEK. However, it can be applied only if $k$ has two children. If this corollary is processed for $k$, the children of $k$ are deallocated, and the depth of $k$ is decreased.

www.redpel.com
+917620593389



$$c_{\mathbf{T}_l'}(k_p) = c_{\mathbf{T}}(k_p) - 1 \qquad c_{\mathbf{T}_r'}(k_p) = c_{\mathbf{T}}(k_p) - 1$$
$$P_{\mathbf{T}_l'}(k_{new}) = P_{\mathbf{T}}(k_l) \times P_{\mathbf{T}}(k) \qquad P_{\mathbf{T}_r'}(k_{new}) = P_{\mathbf{T}}(k) \times P_{\mathbf{T}}(k_r)$$
$$Q_{\mathbf{T}_l'}(k_{new}) = Q_{\mathbf{T}}(k_l) \times Q_{\mathbf{T}}(k) \qquad Q_{\mathbf{T}_r'}(k_{new}) = Q_{\mathbf{T}}(k) \times Q_{\mathbf{T}}(k_r)$$
$$c_{\mathbf{T}_l'}(k_{new}) = c_{\mathbf{T}}(k_l) + c_{\mathbf{T}}(k) \qquad c_{\mathbf{T}_r'}(k_{new}) = c_{\mathbf{T}}(k) + c_{\mathbf{T}}(k_r)$$

Fig. 10. Merging with neighbor key-node

## 5 Proposed Algorithm

### 5.1 Basic Functions

The proposed GKM consists of basic functions for merging or dividing subtrees. Using the theorems and corollaries in the previous section, we introduce the following four basic functions.

**`Splitting KEK`** : According to Theorem 2, we first find the maximum value of $R_{\mathbf{T}}(k_a^d) + R_{\mathbf{T}}(k_b^d)$, where $k_a^d$ and $k_b^d$ are defined as in Theorem 2. Then, for the maximum value, $R_{\mathbf{T}}(k_a^{\hat{d}}) + R_{\mathbf{T}}(k_b^{\hat{d}})$, if the inequality in Theorem 2 is satisfied, we split $k$ into $k_a^{\hat{d}}$ and $k_b^{\hat{d}}$. Otherwise, we do nothing for key $k$. The procedures can be described as following function.

*SplitKey*: key-tree **T**, KEK key-node $k$

```
$pivot \leftarrow 0$;
$R_{max} \leftarrow 0$;
for $i = 1$ to $c_{\mathbf{T}}(k) - 1$ do
    $R \leftarrow \prod_{j=1}^{i} P_{\mathbf{T}}(k_j) + \prod_{j=i+1}^{c_{\mathbf{T}}(k)} P_{\mathbf{T}}(k_j) + i \prod_{j=1}^{i} Q_{\mathbf{T}}(k_j)$
    $+[c_{\mathbf{T}}(k) - i] \prod_{j=i+1}^{c_{\mathbf{T}}(k)} Q_{\mathbf{T}}(k_j)$;
    if $R > R_{max}$ then
        $pivot \leftarrow i$;
        $R_{max} \leftarrow R$;
    end if
end for
if $R_{max} + Q_{\mathbf{T}}(k_p) > R_{\mathbf{T}}(k) + 1$ then
    $\mathcal{K}_{\mathbf{T}}(k_a) \leftarrow \{k_1, k_2, \cdots, k_{pivot}\}$;
    $\mathcal{K}_{\mathbf{T}}(k_b) \leftarrow \{k_{pivot+1}, k_{pivot+2}, \cdots, k_{c_{\mathbf{T}}(k)}\}$;
    $\mathcal{K}_{\mathbf{T}}(k_p) \leftarrow [\mathcal{K}_{\mathbf{T}}(k_p) \backslash \{k\}] \cup \{k_a, k_b\}$;
    return true;
else
    return false;
end if
```

**`Merging 2 KEKs`** : The keys that can be merged with $k$ares $k_l$ and $k_r$, where $k_l$ and $k_r$ are neighbors of $k$, and $k_l \prec k \prec k_r$. Let $k_{can}$ be a candidate for merging with $k$. Then, $k_{can} \in \{k_l, k_r\}$. If there exists no $k_l$, then $k_{can} = k_r$, and if there exists no $k_r$, then $k_{can} = k_l$. If both $k_r$ and $k_l$ exist, we choose between $k_l$ and $k_r$ for $k_{can}$ so that the case for merging $k$ and $k_{can}$ is more optimal than the case for merging $k$ and the other key (Fig. 10). After selecting $k_{can}$, we determine whether we merge $k$ and $k_{can}$. According to Corollary 3, if the inequality is satisfied, we merge $k$ and $k_{can}$. The procedures can be described as following function.

*MergeKey*: key-tree **T**, KEK key-node $k$

```
if $c_{\mathbf{T}}(k_p) < 3$ then
    return false;
end if
// $k_l$, $k_r$ are neighbors of $k$ such that $k_l \prec k \prec k_r$.
if $k_l$ does not exist then
    $k_{can} \leftarrow k_r$;
else if $k_r$ does not exist then
    $k_{can} \leftarrow k_l$;
else
    if $c_{\mathbf{T}}(k)Q_{\mathbf{T}}(k)[Q_{\mathbf{T}}(k_l) - Q_{\mathbf{T}}(k_r)] > [P_{\mathbf{T}}(k_l) - P_{\mathbf{T}}(k_r)][1 - P_{\mathbf{T}}(k)] + [c_{\mathbf{T}}(k_l)Q_{\mathbf{T}}(k_l) - c_{\mathbf{T}}(k_r)Q_{\mathbf{T}}(k_r)][1 - Q_{\mathbf{T}}(k)]$ then
        $k_{can} \leftarrow k_l$;
    else
        $k_{can} \leftarrow k_r$;
    end if
end if
if $R_{\mathbf{T}}(k) + R_{\mathbf{T}}(k_{can}) + Q_{\mathbf{T}}(k_p) < P_{\mathbf{T}}(k) \cdot P_{\mathbf{T}}(k_{can}) + [c_{\mathbf{T}}(k) + c_{\mathbf{T}}(k_{can})] \cdot Q_{\mathbf{T}}(k) \cdot Q_{\mathbf{T}}(k_{can})$ then
    $\mathcal{K}_{\mathbf{T}}(k_{new}) \leftarrow \mathcal{K}_{\mathbf{T}}(k) \cup \mathcal{K}_{\mathbf{T}}(k_{can})$;
    $\mathcal{K}_{\mathbf{T}}(k_p) \leftarrow [\mathcal{K}_{\mathbf{T}}(k_p) \backslash \{k, k_{can}\}] \cup \{k_{new}\}$;
    return true;
else
    return false;
end if
```

**`Creating new child key-nodes`** : Between key $k$ and its child key-nodes, we may insert two new KEKs. According to Theorem 4, we first find the maximum value of $R_{\mathbf{T}}(k_a^d) + R_{\mathbf{T}}(k_b^d)$, where $k_a^d$ and $k_b^d$ are defined as in Theorem 4. Then, for the maximum value $R_{\mathbf{T}}(k_a^{\hat{d}}) + R_{\mathbf{T}}(k_b^{\hat{d}})$, if the inequality in Theorem 4 is satisfied, we create $k$'s two child key-node $k_a^{\hat{d}}$ and $k_b^{\hat{d}}$. Otherwise, we do nothing for key $k$. The procedures can be described as following function.

*CreatChild*: key-tree **T**, TEK or KEK key-node $k$

```
$pivot \leftarrow 0$;
$R_{max} \leftarrow 0$;
for $i = 1$ to $c_{\mathbf{T}}(k) - 1$ do
    $R \leftarrow \prod_{j=1}^{i} P_{\mathbf{T}}(k_j) + \prod_{j=i+1}^{c_{\mathbf{T}}(k)} P_{\mathbf{T}}(k_j) + i \prod_{j=1}^{i} Q_{\mathbf{T}}(k_j)$
    $+[c_{\mathbf{T}}(k) - i] \prod_{j=i+1}^{c_{\mathbf{T}}(k)} Q_{\mathbf{T}}(k_j)$;
    if $R > R_{max}$ then
        $pivot \leftarrow i$;
        $R_{max} \leftarrow R$;
```

www.redpel.com
+917620593389

```
8:     end if
9:  end for
10: if $R_{max} > 2 + [c_{\mathbf{T}}(k) - 2] \cdot Q_{\mathbf{T}}(k)$ then
11:     $\mathcal{K}_{\mathbf{T}}(k_a) \leftarrow \{k_1, k_2, \cdots, k_{pivot}\}$;
12:     $\mathcal{K}_{\mathbf{T}}(k_b) \leftarrow \{k_{pivot+1}, k_{pivot+2}, \cdots, k_{c_{\mathbf{T}}(k)}\}$;
13:     $\mathcal{K}_{\mathbf{T}}(k) \leftarrow \{k_a, k_b\}$;
14:     return true;
15: else
16:     return false;
17: end if
```

**`Deallocating child key-nodes :`** For KEK $k_1$, if the number of siblings of $k_1$ is one, $k_1$ may be merged with $k_2$, where $k_2$ is the sibling key-node. In this case, we deallocate $k_1$ and $k_2$, and we directly connect the parent and children. According to Corollary 5, if the inequality is satisfied, we deallocate $k_1$ and $k_2$. The procedures can be described as following function.

*DeallChild*: key-tree **T**, TEK or KEK key-node $k$

```
1:  if $c_{\mathbf{T}}(k) > 2$ then
2:      return false;
3:  end if
    $//\mathcal{K}_{\mathbf{T}}(k) = \{k_1, k_2\}$.
4:  if $R_{\mathbf{T}}(k_1) + R_{\mathbf{T}}(k_2) < 2 + [c_{\mathbf{T}}(k) - 2] \cdot Q_{\mathbf{T}}(k)$ then
5:      $\mathcal{K}_{\mathbf{T}}(k) \leftarrow [\mathcal{K}_{\mathbf{T}}(k_1) \cup \mathcal{K}_{\mathbf{T}}(k_1)]$;
6:      Deallocate $k_1$ and $k_2$;
7:      return true;
8:  else
9:      return false;
10: end if
```

## 5.2 GKM Framework

In this subsection, we propose a new GKM framework consisting of algorithms for initial construction of a key-tree, and for optimizing the key-tree after the group membership changes.

### 5.2.1 Initial Tree Construction

In this subsection, we describe the algorithm for generating a key-tree from the given users' information. Algorithm 1 is the process for the initial key-tree construction. The input is the information about the group of users, and the users are sorted in increasing order of leaving probability. The output is a key-tree **T**. This algorithm consists mainly of the recursive functions of *SplitKey* or *CreateChild*. It works as follows. First, generate a tree whose height is 1, where the root is a TEK, and the leaf nodes are user-nodes. Then, because there is no KEK key-node, the process operates creating two child key-nodes function. If the function returns true, the recursive function is launched. If it returns false, the algorithm ends.

The detailed process of the recursive function is as follows. For a KEK $k$, the process determines to apply *SplitKey* or *CreateChild*. The criterion for the decision is the lower value of $\mathcal{M}(\mathbf{T})$. If the recursive function returns true, **T** becomes more efficient, and it could

**Algorithm 1** Algorithm for Initial Tree Construction

```
Inputs : User $u_1$, $u_2$, $\cdots$, $u_N$, where $p(u_1) \leq p(u_2) \leq \cdots \leq p(u_N)$
Output : Key-tree $\mathbf{T}$
1:  Allocate TEK in $\mathbf{T}$;
2:  $\mathcal{K}_{\mathbf{T}}(\text{TEK}) \leftarrow \{k_1, k_2, \cdots, k_N\}$;
3:  if CreateChild($\mathbf{T}$,TEK) = true then
4:      for all $k \in \mathcal{K}_{\mathbf{T}}(\text{TEK})$ do
5:          makeTree($k$);
6:      end for
7:  end if
 Function makeTree: key-node $k$
8:  if $[c_{\mathbf{T}}(k) - 1] \cdot Q_{\mathbf{T}}(k) + 1 > R_{\mathbf{T}}(k)$ then
9:      if SplitKey($\mathbf{T}, k$) = true then
10:         for all $k_{child} \in \mathcal{K}_{\mathbf{T}}(k)$ do
11:             makeTree($k_{child}$);
12:         end for
13:     end if
14: else
15:     if CreateChild($\mathbf{T}, k$) = true then
16:         for all $k_{child} \in \mathcal{K}_{\mathbf{T}}(k)$ do
17:             makeTree($k_{child}$);
18:         end for
19:     end if
20: end if
```

become much more efficient yet again. If it returns false, a subtree of **T** of which $k$ is the root is itself the most efficient subtree. Hence, if the basic algorithm returns true, then this recursive function continues, and if the algorithm returns false, the recursive function stops.

### 5.2.2 Optimization After the Group Membership Changes

When the group membership changes, the key-tree can be no longer optimal. In this subsection, we propose another algorithm that maintains the optimality of the key-tree when the group membership changes.

The proposed algorithm has two parts: removing and placing users and re-shaping the structure of the key-tree. For the first part, the algorithm removes the leaving users and places newly joined users by considering the order of users' leaving probabilities. Next, the algorithm marks the KEK key-nodes that are parents of the IK key-nodes of leaving users or joining users. If a marked KEK key-node has child key-nodes for IKs of only leaving users, the node is marked as $-1$, for only joining users, the node is marked as 1, and for both leaving and joining users, the node is marked as 0. Marking it as 1 means that the number of child key-nodes of the marked node increases, and marking it as $-1$ means that on the number decreases.

The second part consists of bottom-up iterations for the depth of the key-tree. Each iteration consists of two parts again: processing the node and marking the parent node. For each iteration, the algorithm finds the

www.redpel.com
+917620593389

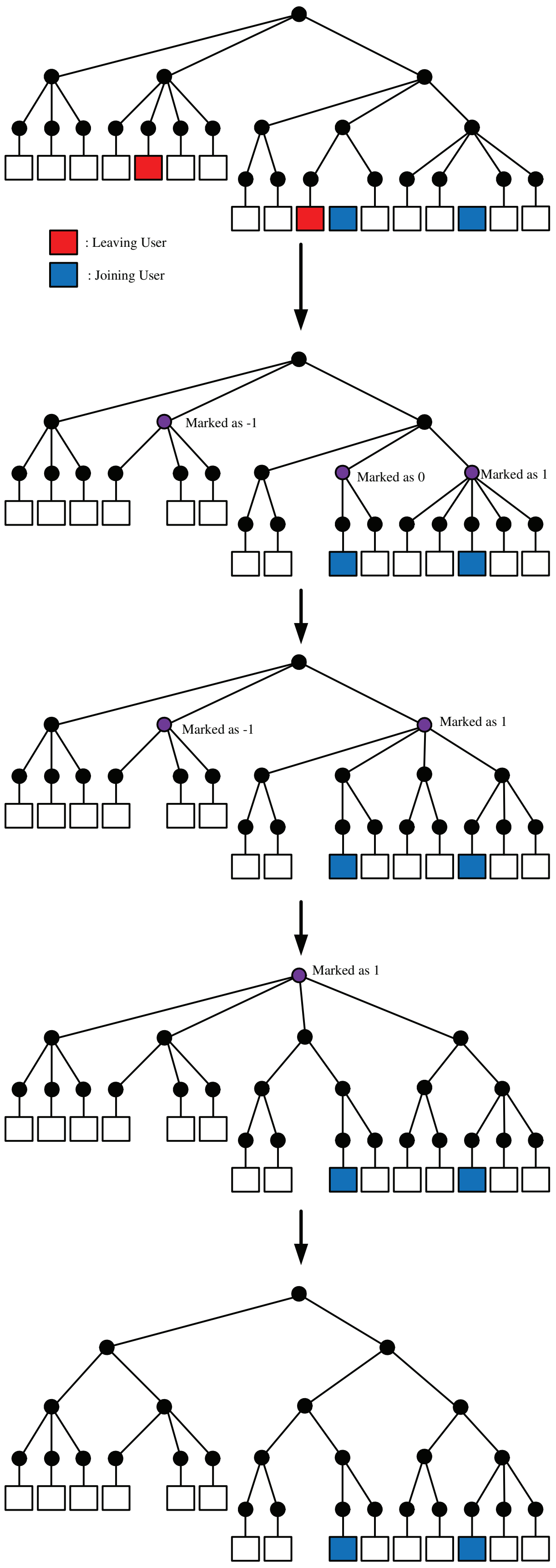


Fig. 11. Example of the Updating Algorithm

**Algorithm 2** Algorithm for Group Membership Changes

```
Input : key-tree $\mathbf{T}$, Joining Users $u_j^1, u_j^2, \cdots, u_j^\lambda$,
        Leaving Users $u_l^1, u_l^2, \cdots, u_l^\mu$
Output : Modified key-tree $\mathbf{T}'$
1: Set the joining users in and remove the leaving users
   from $\mathbf{T}$ in order of the users' leaving probabilities;
2: for $\delta$ = depth of $\mathbf{T} - 1$ to 1 do
3:   for all KEK $k$ in depth $\delta$ of $\mathbf{T}$ do
4:     if $c_\mathrm{T}(k)$ increases then
5:       Mark $k$ as 1;
6:       if Splitting $k$ is more optimal than creating $k$'s
         new children then
7:         SplitKey($\mathbf{T}$,$k$);
8:       else
9:         CreateChild($\mathbf{T}$,$k$);
10:      end if
11:    else if $c_\mathrm{T}(k)$ decreases then
12:      Mark $k$ as $-1$;
13:      if $c_\mathrm{T}(k_p) > 2$ then
14:        MergeKey($\mathbf{T}$,$k_p$);
15:      else
16:        DeallChild($\mathbf{T}$,$k_p$);
17:      end if
18:    else
19:      Mark $k$ as 0;
20:    end if
21:  end for
22: end for
23: Update the leaving probability of the leaving users;
```

marked key-nodes at the corresponding depth. The algorithm considers only the key-nodes marked as described above. If the node is marked as 1, either the function for splitting node or that for creating child key-nodes is executed. When the function returns true, the splitting or creating operation was successful, and the parent of the current node is marked as 1. If the node is marked as $-1$, either the function for merging nodes or that for deallocating child key-nodes is executed. When the function returns true, the merging or deallocating operation was successful, and the parent of the current node is marked as $-1$. Finally if the node is marked as 0, the algorithm chooses a function from among the four basic algorithms to make the current node optimal. If splitting or creating is executed, the parent key-node is marked as 1, and if merging or deallocating is executed, it is marked as $-1$.

After regenerating the key tree, the algorithm finally updates the leaving probabilities of the leaving users. The updating process is as follows. For a leaving user $u_l^i$, let $I(u_l^i)$ and $G(u_l^i)$ be the number of rekey intervals in which the user subscribes and the number of leaving the group, respectively. After finishing Algorithm 2, $G(u_l^i)$ increases by 1, and $I(u_l^i)$ increases by the number of rekey intervals in which $u_l^i$ subscribes in the current

www.redpel.com
+917620593389

group. Then, $p(u_l^i)$ is updated to $\frac{G(u_l^i)}{I(u_l^i)}$.

## 5.3 Computational Cost

Now, we analyze the computational cost for the proposed algorithms. Because the algorithms are composed of the basic functions which are introduced in 5.1, we first investigate the complexity of the basic functions, after that, we discuss the proposed algorithm.

### 5.3.1 *For Basic Functions*

We proposed the four basic functions. However, functions $MergeKey$ and $DeallChild$ do not have any iterations, so that the computational complexity is constant. Therefore we analyze the complexity of $SplitKey$ and $CreateChild$.

The two functions are composed of the same procedures except for line 10. Also, the computational costs for the functions are mostly emerged in an iteration of lines 3-9, respectively. Each function is composed of $c_{\mathbf{T}}(k) - 1$ iterations, and the computational complexity for each iteration is $O(c_{\mathbf{T}}(k))$ due to line 4. Hence, the computational complexity for each function is $O(c_{\mathbf{T}}(k)^2)$.

### 5.3.2 *For Proposed Algorithms*

First, we discuss the initial tree construction algorithm. As Algorithm 1, the process is composed of a recursive function $makeTree(k)$. The computational costs for line 9 and 15 are $O(c_{\mathbf{T}}(k)^2)$, respectively. Also, $c_{\mathbf{T}}(k) = |\mathcal{U}_{\mathbf{T}}(k)|$ since line 9 or 15 is processed when $k$ and its child nodes compose a flat tree. Therefore, the computational costs for line 9 and 15 are $O(|\mathcal{U}_{\mathbf{T}}(k)|^2)$, respectively. Furthermore, function $makeTree$ is processed for a TEK and every KEK, in turn, the computational cost for Algorithm 1 is

$$O\left(\sum_{k \in \mathcal{K}_{\mathbf{T}}^T \cup \mathcal{K}_{\mathbf{T}}^K} |\mathcal{U}_{\mathbf{T}}(k)|^2\right).$$

Now, we analyze the computational cost for algorithm for optimization after the group membership changes. As Algorithm 2, the basic functions are processed for only the TEK or KEKs whose descendants are modified. The number of such a TEK or KEKs is not greater than the number of entire TEK and KEKs. Furthermore, complexity for each basic function is not greater than $O(c_{\mathbf{T}}(k)^2)$, where $k$ is a current key-node of procedure in lines 4-20. Hence, the computational cost for Algorithm 2 is not greater than the following value:

$$O\left(\sum_{k \in \mathcal{K}_{\mathbf{T}}^T \cup \mathcal{K}_{\mathbf{T}}^K} c_{\mathbf{T}}(k)^2\right).$$

## 6 Simulations

To evaluate the performance of the framework, we conducted simulations of the proposed approach and compared it to the existing one. The simulations consists of two parts: one for the building algorithm and one for dynamic scenarios of users' joining and leaving. The results are compared in terms of the average processing time and the average size of the rekeying messages.

| Group Size | 4-ary BR($\mathbf{T}_4$) | Proposed Key-Tree($\mathbf{T}$) |
|---|---|---|
| 16 | 0.594ms, 3.54 | 23.5ms, 2.56 |
| 64 | 0.813ms, $1.80 \times 10$ | 29.8ms, $1.35 \times 10$ |
| 256 | 3.98ms, $7.61 \times 10$ | 42.5ms, $5.61 \times 10$ |
| 1024 | 7.12ms, $3.08 \times 10^2$ | 59.7ms, $2.28 \times 10^2$ |
| 4096 | 13.2ms, $1.24 \times 10^3$ | 85.1ms, $9.13 \times 10^2$ |
| 16384 | 29.6ms, $4.96 \times 10^3$ | 185ms, $3.65 \times 10^3$ |
| 65536 | 103ms, $1.98 \times 10^4$ | 592ms, $1.45 \times 10^4$ |

TABLE 2
Results for Initial Construction of Tree Structure(Processing Time(ms) and Average size of Rekeying Messages)

## 6.1 Building the Tree Structure

We simulate Algorithm 1 and investigate its performance. For simulation model, we assume that there are $N$ users and their leaving probabilities are given, and construct a new key-tree $\mathbf{T}$ by using the algorithm. For comparison, we choose a 4-ary balanced and complete BR $\mathbf{T}_4$, which has been proven to be the optimal BR. Specifically, we set the number of users, $N$, to a power of 4, because a full-4-ary BR whose height is $h$ has $4^h$ users.

We measure the processing time(ms) for constructing the key-tree, and calculate the average number of rekeying messages $\mathcal{M}(\mathbf{T})$ when the number of subscribers($N$) is given. The simulation works as follows. We first set $N$ users and their leaving probabilities. The leaving probabilities are determined randomly with an average of 0.9. Second, we build the 4-ary balanced and complete BR, $\mathbf{T}_4$. Third, we build a key-tree $\mathbf{T}$, by using Algorithm 1. We measure the processing time and calculate $\mathcal{M}(\mathbf{T}_4)$ and $\mathcal{M}(\mathbf{T})$ by employing equation (7). We iterate the above procedures 100 times, and derive the average processing times and average value of $\mathcal{M}(\mathbf{T})$ and $\mathcal{M}(\mathbf{T}_4)$.

Table 2 compares the average processing time and size of the keying messages for $4^h$ users, where $h$ is an integer, and $1 \leq h \leq 7$. The processing time for the proposed algorithm is longer than one for generating the 4-ary balanced and complete BR, but the times are absolutely short, i.e., the proposed algorithm terminates within a resonable time. Also, for the computational cost, the average value of $\mathcal{M}(\mathbf{T})$ is $30 \sim 40\%$ as small as $\mathcal{M}(\mathbf{T}_4)$.

## 6.2 For Dynamic Scenarios of Users' Joining and Leaving

We simulate Algorithm 2 with group dynamics to maintain the optimality of the key tree. For simulation model,

www.redpel.com
+917620593389

| Group Size | 4-ary BR($\mathbf{T}_4$) | Proposed Key-Tree($\mathbf{T}$) |
|---|---|---|
| 16 | 0.120ms | 13.6ms |
| 64 | 0.369ms | 15.2ms |
| 256 | 0.865ms | 20.8ms |
| 1024 | 2.82ms | 28.6ms |
| 4096 | 6.13ms | 52.9ms |
| 16384 | 21.4ms | 113ms |
| 65536 | 80.1ms | 381ms |

TABLE 3
Average Processing Time for Optimizing Tree Structure

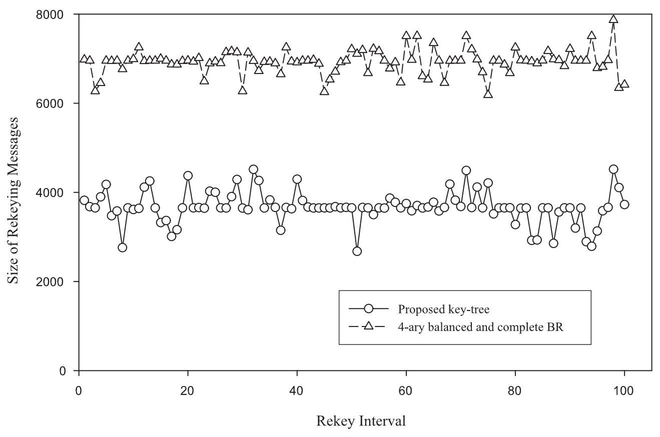


Fig. 12. Result of Simulation for Optimizing Tree Structure

we assume that there are $N$ subscribers, and their leaving probabilities are given. We generate two key-trees $\mathbf{T}$ and $\mathbf{T}_4$ by using the construction processes in 6.1. And then, we simulate the subscribers' joining and leaving freely, as in real-life situations. In every rekey interval, group rekeying is executed, for $\mathbf{T}$, tree structure modification is also executed.

To observe the computational and communication cost, we measure the processing time and the number of rekeying messages in every rekey interval. The simulation works as follows. We first set $16384$ users, determine their leaving probabilities randomly, and build key-trees $\mathbf{T}$ and $\mathbf{T}_4$, where $\mathbf{T}$ is made by using Algorithm 1 and $\mathbf{T}_4$ is made as 4-ary balanced and complete BR. After that, we simulate that each user leaves or joins the group according to one's leaving probability. For each rekey interval, we check the joining and leaving users and calculate the size of rekeying messages for $\mathbf{T}$ and $\mathbf{T}_4$. For $\mathbf{T}$, we also apply Algorithm 2 to maintain optimality.

Table 3 indicates the average processing times. Although the time for proposed scheme is larger than the original $4$-ary BR, Algorithm 2 is terminated within a reasonable time. Figure 3 compares the size of rekeying messages for our proposed key-tree and a 4-ary BR. The plot for the proposed key-tree shows that optimality is preserved in most rekey intervals. Also, for all rekey intervals, the average size of the rekeying messages for the proposed one is 45% less than that for the 4-ary BR. Hence, we confirm that Algorithm 2 maintains the key-tree's optimality.

## 7 Conclusion

This paper proposes a new GKM framework for the environment with dynamic mobile subscribers. Unlike other tree-based GKM schemes, ours assumes no special conditions: users have unequal leaving probabilities, and the key-tree is unbalanced and incomplete. We first compute the exact communication cost for the general key-tree. On the basis of the accurate communication cost, we propose two algorithms: one for initial key-tree construction, and the other for optimizing the key-tree when the group membership changes. The first algorithm generates a more optimal key-tree structure than that in previous works, and the second algorithm maintains its optimality. Furthermore, instead of using the special trees, i.e., balanced and complete key-trees, we employ a general tree structure, so our GKM can support any number of users. In our first simulation, we showed that the average communication overhead of a key-tree made by our initial key-tree construction algorithm for group rekeying is $30 \sim 40\%$ smaller than that of a 4-ary balanced and complete key-tree, which was identified as the most optimal tree in previous works. In our second simulation, we confirmed that the second algorithm maintains the optimality of the key-tree by showing that during group membership changes, the average communication cost for our generated key-tree is $45\%$ less than that for the 4-ary balanced and complete key-tree. If our proposed GKM is applied to multicasting, GKM would be more cost-effective than using existing key-trees.

## References

[1] Sanjoy Paul. *Multicasting on the Internet and Its Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[2] M. Park, Y. Park, H. Jeong, and S. Seo. Secure multiple multicast services in wireless networks. *Mobile Computing, IEEE Transactions on*, PP(99):1, 2012.

[3] H. Harney and C. Muckenhirn. Group key management protocol (gkmp) specification, 1997.

[4] H. Harney and C. Muckenhirn. Group key management protocol (gkmp) architecture, 1997.

[5] Jack Snoeyink, Subhash Suri, and George Varghese. A lower bound for multicast key distribution. *Comput. Netw.*, 47(3):429–441, February 2005.

[6] Min-Ho Park, Young-Hoon Park, and Seung-Woo Seo. A cell-based decentralized key management scheme for secure multicast in mobile cellular networks. In *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, pages 1 –6, may 2010.

[7] Jen-Chiun Lin, Feipei Lai, and Hung-Chang Lee. Efficient group key management protocol with one-way key derivation. In *Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, LCN '05, pages 336–343, Washington, DC, USA, 2005. IEEE Computer Society.

[8] Wen Tao Zhu. Optimizing the tree structure in secure multicast key management. *Communications Letters, IEEE*, 9(5):477 – 479, may 2005.

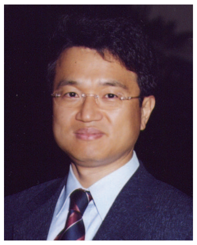
www.redpel.com
+917620593389

[9] Jun Sik Lee, Ju Hyung Son, Young Hoon Park, and Seung Woo Seo. Optimal level-homogeneous tree structure for logical key hierarchy. In *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, pages 677 –681, jan. 2008.
[10] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Trans. Netw.*, 8(1):16–30, February 2000.
[11] Sandro Rafaeli and David Hutchison. A survey of key management for secure group communication. *ACM Comput. Surv.*, 35(3):309–329, September 2003.
[12] Yan Sun, Wade Trappe, and K. J. R. Liu. A scalable multicast key management scheme for heterogeneous wireless networks. *IEEE/ACM Trans. Netw.*, 12(4):653–666, August 2004.
[13] V. Kondratieva and Seung-Woo Seo. Optimized hash tree for authentication in sensor networks. *Communications Letters, IEEE*, 11(2):149 –151, feb. 2007.
[14] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: a taxonomy and some efficient constructions. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 708 –716 vol.2, mar 1999.
[15] Germano Caronni, Marcel Waldvogel, Dan Sun, and Bernhard Plattner. Efficient security for large and dynamic multicast groups. In *Proceedings of the 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE '98, pages 376–383, Washington, DC, USA, 1998. IEEE Computer Society.
[16] M. J. Moyer, J. R. Rao, and P. Rohatgi. A survey of security issues in multicast communications. *Netwrk. Mag. of Global Internetwkg.*, 13(6):12–23, November 1999.
[17] W. Trappe, Jie Song, R. Poovendran, and K. J. R. Liu. Key distribution for secure multimedia multicasts via data embedding. In *Proceedings of the Acoustics, Speech, and Signal Processing, 2001. on IEEE International Conference - Volume 03*, ICASSP '01, pages 1449–1452, Washington, DC, USA, 2001. IEEE Computer Society.
[18] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures, 1999.
[19] Alan T. Sherman and David A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Trans. Softw. Eng.*, 29(5):444–458, May 2003.
[20] Ran Canetti, Tal Malkin, and Kobbi Nissim. Efficient communication-storage tradeoffs for multicast encryption. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 459–474, Berlin, Heidelberg, 1999. Springer-Verlag.
[21] Daniele Micciancio and Saurabh Panjwani. Optimal communication complexity of generic multicast key distribution. *IEEE/ACM Trans. Netw.*, 16(4):803–813, August 2008.
[22] Xiaozhou Steve Li, Yang Richard Yang, Mohamed G. Gouda, and Simon S. Lam. Batch rekeying for secure group communications. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 525–534, New York, NY, USA, 2001. ACM.
[23] Dong-Hyun Je, Jun-Sik Lee, Yongsuk Park, and Seung-Woo Seo. Computation-and-storage-efficient key tree management protocol for secure multicast communications. *Comput. Commun.*, 33(2):136–148, February 2010.
[24] Yang Richard Yang, X. Steve Li, X. Brian Zhang, and Simon S. Lam. Reliable group rekeying: a performance analysis. *SIGCOMM Comput. Commun. Rev.*, 31(4):27–38, August 2001.
[25] Wee Hock Desmond Ng, Michael Howarth, Zhili Sun, and Haitham Cruickshank. Dynamic balanced key tree management for secure multicast communications. *IEEE Trans. Comput.*, 56(5):590–605, May 2007.
[26] Balanced batch lkh: New proposal, implementation and performance evaluation. In *Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, ISCC '03, pages 815–, Washington, DC, USA, 2003. IEEE Computer Society.
[27] Yang Ji and Seung-Woo Seo. Optimizing the batch mode of group rekeying: lower bound and new protocols. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 1846–1854, Piscataway, NJ, USA, 2010. IEEE Press.

**Young-Hoon Park** received B.S., M.S., and Ph.D degrees from Seoul National University, Seoul, South Korea in 2006, 2008, and 2013 respectively. He was a researcher of the Institute of New Media and Communications in Seoul National University. He has received scholarship from Seoul National University, Brain Korea 21 and Samsung Electronics. He is a senior engineer for Cloud Solution Team of Digital Media & Communication R & D Center in Samsung Electronics. His current research areas include network security, privacy, cryptography, secure protocol designing, and system optimization.

**Dong-Hyun Je** is currently working as a senior engineer of telecommunication-systems business in Samsung Electronics, designing LTE system architecture. He received his PH.D. and B.S. degree in Electrical Engineering from Seoul National University, Seoul, Republic of Korea, in 2012 and 2006, respectively. He was a researcher of the Institute of New Media and Communications, and Intelligent Vehicle IT (IVIT) Research Center funded by Korean Government and Automotive Industries in Seoul National University. He has received scholarship from Seoul National University, Brain Korea 21 and Samsung Electronics. His current research interests include wireless networks, vehicular communication networks and computer & network security.

**Min-Ho Park** is an assistant professor at School of Electronic Engineering, Soongsil University, Seoul, Korea. He received the B.S. and M.S. degrees in electronics engineering from Korea University in 2000 and 2002, respectively. He worked at Samsung Electronics from 2002 to 2004. He then received the Ph.D. degree with the School of Electrical Engineering and Computer Science from Seoul National University, Seoul, Korea in 2010. As a postdoctoral researcher, he had worked at Carnegie Mellon University for two years since 2011. Before the postdoctoral researcher at CMU, he was a senior engineer for 3GPP LTE S/W Development Group of Samsung Electronics. His current research interests include wireless networks, vehicular communication networks, network security and cloud computing.



**Seung-Woo Seo** is the professor of Electrical Engineering in Seoul National University, Seoul and Director of Intelligent Vehicle IT (IVIT) Research Center funded by Korean Government and Automotive Industries. He received his Ph.D. from Pennsylvania State University, University Park, USA, and B.S. and M.S. degrees from Seoul National University, Seoul, Korea, and all in Electrical Engineering. He was with the Faculty of the Department of Computer Science and Engineering, Pennsylvania State University, and served as a Member of the Research Staff in the Department of Electrical Engineering in Princeton University, Princeton, NJ. In 1996, he joined the Faculty of the School of Electrical Engineering, Institute of New Media and Communications and Automation and Systems Research Institute in Seoul National University. He has served as Chair or a Committee Member in various international conferences and workshops including INFOCOM, GLOBECOM, PIMRC, VTC, MobiSec, Vitae, etc. He also served for five years as a Director of the Information Security Center in Seoul National University. His research areas include vehicular electronics for intelligent vehicles, communication networks, computer & network security, and system optimization.

www.redpel.com
+917620593389