Theoretical Computer Science 407 (2008) 511–523

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Lower bounds and new constructions on secure group communication schemes

Scott C.-H. Huang [a,*], Frances Yao [a,1], Minming Li [a,2], Weili Wu [b,3]

[a] Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong

[b] Department of Computer Science, University of Texas at Dallas, Mail Station EC 31, 2601 North Floyd Road, Richardson, TX 75083, United States

ARTICLE INFO

Article history:
Received 22 May 2008
Accepted 4 August 2008
Communicated by D.-Z. Du

Keywords:
Secure group communication

ABSTRACT

This paper presents both the theoretical and practical aspects of secure group communication schemes. We pointed out that multiple revocation is a fundamentally time-consuming task in secure group communication, by establishing lower bounds for broadcast encryption and group key distribution schemes. We showed that they are $O(n)$ for BE and $O(n/m)$ for GKD respectively, where m is storage requirement and n is the number of users. Thus, they are clearly far more costly than the ideal log bound. In practice, we designed a new broadcast encryption scheme RBE that actually achieves these lower bounds. RBE is shown to outperform most efficient BE schemes in mass revocation. We discuss the influence of *join* as well as the feasibility of adding it in BE schemes by means of performing full updating or overprovisioning.

1. Introduction

Secure group communication, one of the major problems in wireless network security, is gaining popularity over the last decade. Due to wireless communication's emerging applications and its nature of insecurity, the need for confidentiality, authenticity, as well as other security features was increased far more than ever before. In all of those needs or requirements, the fundamental problem: "*How to distribute keys securely*" has to be dealt with immediately, since nearly all types of security-related functionalities are involved in sharing cryptographic keys. How to distribute those keys over insecure wireless channels is quite a challenging problem.

There are several fundamentally different approaches to deal with key management. Key pre-distribution schemes refer to methods whereby a trusted authority (TA) distributes secret information to users, such that only privileged users can get the group key (by performing computation or decryption). Contributory group key agreement schemes require that each member contribute an equal share of the group key and multi-party Diffie–Hellman Key Exchange is an example of this. Group key management schemes use a trusted third party (TTP) to perform all bookkeeping tasks. In this category, there are flat schemes, in which only one group key is maintained and used by everybody, and hierarchical ones, in which all communications are passed through agents. From a different point of view, there are centralized schemes, as well as decentralized ones. In regard to their functionalities, we can further divide flat schemes into two types: *broadcast encryption* (BE), in which *join* is not provided , and *group key distribution* (GKD), in which both *join* and *leave* are provided.

* Corresponding author. Tel.: +852 2788 8504; fax: +852 2788 8614.
E-mail addresses: shuang@cityu.edu.hk, shuang@cs.cityu.edu.hk (S.C.-H. Huang), csfyao@cityu.edu.hk (F. Yao), minmli@cs.cityu.edu.hk (M. Li), weiliwu@utdallas.edu (W. Wu).
1 Tel.: +852 2194 2907; fax: +852 2788 8614.
2 Tel.: +852 2788 9538; fax: +852 2788 8614.
3 Tel.: +1 972 883 2194.

0304-3975/$ – see front matter © 2008 Published by Elsevier B.V.
doi:10.1016/j.tcs.2008.08.024

In this paper, we focus on flat group key management schemes. We found that, in such schemes, *multiple join/leave* is intrinsically a time-consuming operation. We found the lower bounds on multiple join/leave in both type of schemes in terms of re-keying cost, and proved that in group key distribution schemes it is at least $\Omega(n)$ (linear to the number of nodes), and in any broadcast encryption schemes it is at least $\Omega(n/m)$ (where m is the size of memory). We then designed RBE, the *Randomized Broadcast Encryption* scheme that actually achieves this bound. We showed that RBE outperforms nearly all other BE schemes in the case of mass revocation, while preserving the advantage of redundancy, a property crucial to wireless communications but most efficient BE schemes lack. At the end of this paper, we analyzed the nuance of *join*, a decisive factor that greatly influences the overall performance of all secure group communication schemes.

The rest of this paper is organized as follows. We introduce broadcast encryption and group key distribution schemes in Section 2 and present the lower bounds of group key distribution and broadcast encryption in Sections 3 and 4, respectively. Section 5 is about our randomized broadcast encryption scheme that meets the lower bounds. We also discuss in detail the influence of join in Section 6. Finally we present the related works of this paper in Section 7, and conclude this discussion in Section 8.

2. Broadcast encryption and group key distribution

2.1. Overview

Broadcast encryption and group key distribution schemes play a central role in Secure group communication protocols. Without facilitating the *join* operation, broadcast encryption schemes find themselves less restricted to *key updating*, thus achieving better performance in doing *multiple revocation*. Group key distribution schemes, on the other hand, provide both join and revocation, thus guaranteeing both *forward* and *backward secrecy*.

2.1.1. Broadcast encryption schemes

The term *broadcast encryption* schemes, first coined by Fiat and Naor [12] in 1993, can be actually viewed as *revocation schemes*. In such schemes, a trusted authority called the *Group Controller* (GC) is in charge of bookkeeping for a pre-defined set of users. In each session, GC will select a *Traffic Encryption key* (TEK), distribute it to a subset of users called the *privileged users*, and then encrypt all communications with it. The distribution of TEK will certainly be secure against eavesdropping. In other words, the broadcast of TEK will be encrypted, and only those privileged users who hold some secret information will be able to decrypt it. In such settings, only those in the pre-defined set are eligible to be a privileged user. No other users are allowed to join the group later. There thus exists a set of *prospective* privileged users, which cannot be changed. This strong assumption makes it possible to facilitate a rather hard operation, namely *multiple revocation*, which was previously thought unfeasible in group key distribution schemes.

2.1.2. Group key distribution schemes

Group key distribution Schemes provide both *join* and *revocation* operations. There does not exist a set of pre-defined *prospective users*, and any user can join the group at any time. Such a nice feature makes them more flexible and therefore best suit most applications. Like broadcast encryption schemes, group key distribution schemes have a Group Controller, and each user carries some secret information. When membership changes, GC will choose a new TEK, encrypt it using users' secret information, and then broadcast it. Only those who remain in the group can use their secret information to decrypt the message and get the TEK update. However, unlike broadcast encryption schemes, when a user gets revoked, their secret information will not be used again, and whoever holds same piece of the information will get updated. It is just because of this extra updating are group key distribution schemes capable of joining new members. And it is just because of this, as well, they are not efficient for revoking multiple members, since the extra updating of multiple users is usually too slow to be practical.

2.1.3. Multiple revocation

As mentioned earlier, group key distribution schemes cannot handle multiple revocation efficiently. Broadcast encryption schemes, on the other hand, can manage it better because there is no overhead in providing *join*. *Multiple Revocation*, a fundamentally difficult problem, may be intrinsically incompatible with *join*, and the goal of this paper as well as our main motivation is therefore centered around this problem.

3. Re-keying cost of group key distribution schemes

The goal of this section is to establish a lower bound for re-keying cost of GKD schemes. The result shows that this bound is actually quite large, as we proved that it is at least linear to the number of users.

3.1. System model

Let there be a group of users and a group controller (GC). All users and GC share a TEK. When someone wants to broadcast a message to the entire group, they will use TEK to encrypt the message to ensure confidentiality. When there is a user that

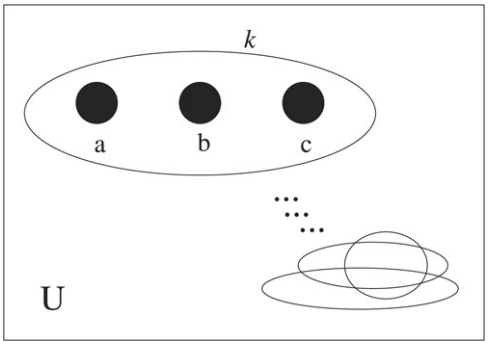

Fig. 1. $k \in \mathcal{A}$ meaning k is shared by a, b, c.

leaves the group (or many users leaving the group), TEK must be updated to ensure confidentiality. Updating TEK in case of membership change is called *revocation*. To facilitate revocation, we can use key encryption keys (KEKs), shared by certain users and GC, to encrypt TEK update messages. In addition to updating TEK, leaving users also hold certain KEKs that need to be updated as well. A *re-keying message* contains updated TEK and KEKs and is itself encrypted with an appropriate KEK, such that only legitimate (remaining) users can decrypt it and get the updates. A *key assignment* is the way KEKs are shared, which can be modeled using set-theoretic notations in the next paragraph.

3.2. Problem description

Let $U = \{u_1, u_2, \ldots, u_n\}$ be a group of n users. We use a collection of subsets of U to represent the key-sharing relationship among users. If $v_1, v_2, \ldots, v_j$ shares a common KEK, we add the subset $\{v_1, v_2, \ldots, v_j\}$ in the collection. We can thus use this way to completely formulate the key-sharing relationship of any users in U.

> The key-sharing relationship among users in U can be modeled by a collection $\mathcal{A}$ of subsets in U, defined as follows. $X \in \mathcal{A} \iff$ all users in X share a common KEK.

Fig. 1 shows that we can use subsets to represent KEKs. In this example, k is shared by a, b, c, represented by $k \in \mathcal{A}$.

TEK must be updated if there is membership change. Let $R \subset U$ be the set of leaving users, then GC must update TEK and broadcast it to all users in $U - R$. A key assignment must be able to support revocation for all possible R. In particular, we observe the following.

- If $|R| = n - 1$ then $U - R$ is a one-element subset. Therefore, $\mathcal{A}$ must contain all one-element subsets
- If $R = \emptyset$ then $U - R = U$. Therefore, $\mathcal{A}$ must contain U itself. This particular set can be regarded as the TEK.

Based on this observation, we can define the idea of key assignment as follows.

Definition 3.1 (*Key Assignment*). A key assignment is a collection $\mathcal{A}$ of subsets in U that contains U itself and all one-element subsets.

When R is removed from U, GC must update (or re-key) TEK and invalid KEKs by using appropriate KEKs. Namely, GC needs to generate new keys and encrypt it with certain KEKs. The re-keying cost is presented as follows (the formal definition will be introduced in the next section):

- *Re-keying cost of TEK*: This is actually the smallest number of valid KEKs that covers the remaining nodes (i.e. $U - R$). If all valid KEKs are one-element subsets, this cost is obviously the size of remaining nodes (i.e. $|U - R|$). We will use this to derive a lower bound at the end of this section.
- *Re-keying cost of KEKs*: Namely the number of KEKs used to encrypt invalid KEKs. This cost is hard to estimate, but obviously it is at least as large as the number of invalid KEKs. We will use this property to derive a lower bound at the end of this section as well.

In this section, we focus on the re-keying cost of group key distribution schemes, which includes both cases. In the next section, namely the broadcast encryption schemes, KEKs are assumed not to be updated at all, so its re-keying cost only involves TEK. Now, we actually have not formally defined the re-keying cost, yet. Instead, we only use the two properties as described earlier. These two properties alone are enough for us to derive useful results. The formal definition of re-keying cost will be introduced in the next section. The goal for this section is to prove the following theorem.

Theorem 3.1. *Suppose $\mathcal{A}$ is a key assignment of a group key distribution scheme over U. Let $|U| = n$, then its re-keying cost is at least $\Omega(n)$.*

To prove this theorem, we construct a set of users R for any key assignment $\mathcal{A}$, whose removal will require a linear updating cost.

Definition 3.2. Let $\mathcal{A}$ be a key assignment. A subset of users $H \subset U$ is called a hitting set for $\mathcal{A}$ if we have the following property

$$U_j \cap H \neq \emptyset \quad \forall U_j \in \mathcal{A} \text{ with } |U_j| > 1.$$

The hitting number h of $\mathcal{A}$ is the size of a minimum hitting set for $\mathcal{A}$.

Following this definition, a hitting set H has the property that, if we delete all users in H and any keys containing a user in H, then the remaining users will not share any key. To prove this theorem, we use a marking process to select a set of users R for removal, and show that its associated updating cost will be at least linear to the number of total users.

Marking Algorithm (MA)

Input: key assignment $\mathcal{A}$
Output: a subfamily $\mathcal{S} \subset \mathcal{A}$ with $\mathcal{S} = J$, and subsets *RED*, *BLUE* $\subset U$

1. Let $\mathcal{S} = \emptyset$ and $J = 0$.
   Repeat 2–4:
2. Pick a U_j with $|U_j| > 1$ in $\mathcal{A} - \mathcal{S}$ whose elements are all unmarked. Stop if no such U_j exists.
3. Pick two elements from U_j, mark one element red and the other blue.
4. Add U_j to $\mathcal{S}$ and let $J \leftarrow J + 1$.

Let *RED* and *BLUE* respectively denote the subsets of U that are marked red and blue when MA stops. Let h be the hitting number for $\mathcal{A}$.

Lemma 3.1. *MA satisfies the following properties:*

(i) *Each execution of step* 3 *marks two previously unmarked elements.*
(ii) *When MA stops, every* $U_j \in \mathcal{S}$ *contains at least one red and one blue element.*
(iii) $|RED \cup BLUE|$ *forms a hitting set for* $\mathcal{A}$.

Proof. (i) and (ii) are obvious from the algorithm description. (iii) is true because of the stopping condition in step 2 of MA. ■

Lemma 3.2. *When MA stops, we have*

(i) $h \leq |RED \cup BLUE| = 2J$
(ii) *When RED is removed from U, each* $U_j \in \mathcal{S}$ *becomes strictly smaller but remains nonempty.*

Proof. (i) follows from properties (i) and (iii) of Lemma 3.1, while (ii) is a consequence of property (ii) of Lemma 3.1. ■

Proof of Theorem 3.1. We specify a set $R \subset U$ to be removed, such that its re-keying cost will be $\Omega(n)$. Let h be the hitting number for $\mathcal{A}$.
Case 1: $h \leq 2n/3$. We claim that the cost for updating TEK is at least $n/3$. To prove this, let H be a minimum hitting set and let $R = H$. Since H is a hitting set, when R is removed from U the remaining users will not have any shared key (i.e. the valid KEKs are all one-point subsets). Therefore, the cost for updating TEK is at least $|U - R|$, and $|U - R| = n - h \geq n/3$.
Case 2: $h > 2n/3$. We claim that the cost for updating KEKs is at least $n/3$. To prove this, we execute MA and let $R = RED$. When R is removed, each KEK corresponding to a subset $U_j \in \mathcal{S}$ needs to be updated because of property (ii) of Lemma 3.2. The total number of such KEKs is $J \geq h/2 > n/3$ by Lemma 3.2. Thus the re-keying cost is at least $n/3$ in both cases. ■

4. Re-keying cost of broadcast encryption schemes

The goal of this section is to establish a lower bound for re-keying cost of BE schemes. Different from GKD schemes, in deriving the lower bound the idea of *storage requirement* needs to be taken into consideration. The result we got is quite similar to that of GKD, but the methods used are totally different.

4.1. System model

The system model of broadcast encryption schemes is very similar to that of group key distribution schemes, discussed in Section 3. The only difference is that in case of a revocation, only the TEK are updated. Therefore the re-keying cost will be smaller since it only involves updating TEK.

4.2. Problem description

U, $\mathcal{A}$ are defined the same way as in Section 3. When R is removed from U, GC must update (or re-key) TEK by using other KEKs. Namely, GC needs to generate a new TEK and encrypt it with certain KEKs. The re-keying cost is defined as the

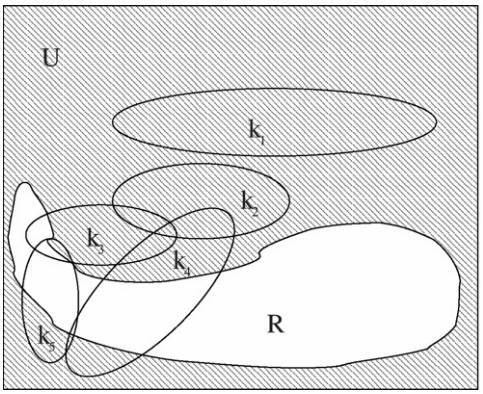

Fig. 2. Key assignment example: only k_1 and k_2 can be used to update TEK. k_3, k_4, k_5 are all invalid.

smallest number of KEKs used to encrypt the message. Note that only the KEKs that completely lie in $U - R$ can be used to do the update, as shown in Fig. 2. The formal definition of re-keying cost is as follows.

Definition 4.1 (*Re-keying Cost*). Suppose $\mathcal{A}$ is a key assignment, and $R \subset U$ is the set of leaving users. Let $\mathcal{A}_R$ be $\{X | X \in \mathcal{A}, X \subset U - R\}$. Then re-keying cost of $\mathcal{A}$ is defined as $\min |\mathcal{C}_R|$, where $\mathcal{C}_R \subset \mathcal{A}_R$ is a covering of $U - R$ (i.e. $\bigcup_{X \in \mathcal{C}_R} X \supset U - R$) and the minimum is taken over all possible $\mathcal{C}_R$'s.

In Section 3, we proved Theorem 3.1, which says the re-keying cost of any group key distribution schemes is at least of linear bound (with respect to the number of users). Note that, in that theorem, no storage requirement is involved. However, in broadcast encryption schemes, taking storage into consideration is a must. If storage is unlimited, then the re-keying cost can be constant and the linear bound is no longer a lower bound. Consider the following example.

Let $\mathcal{A}$ be 2^U, the power set of U. Then the re-keying cost for any R leaving U is $O(1)$.

This example tells us that we need to formalize the notion of storage requirement in order to derive a lower bound for broadcast encryption schemes. It is as follows.

Definition 4.2 (*Storage Requirement*). Suppose $\mathcal{A}$ is a key assignment. At any point $x \in U$ we define the notion of overlap as follows.

$$Overlap(\mathcal{A}, x) = \left|\{S | S \in \mathcal{A}, S \ni x\}\right|$$

where $\left|\cdots\right|$ represents the size of the set in between. The storage (memory) requirement is defined as the maximal overlap of all points as follows.

$$Storage(\mathcal{A}) = \max_{x \in U} Overlap(\mathcal{A}, x).$$

4.3. Lower bound

Theorem 4.1. *Suppose $\mathcal{A}$ is a key assignment for U with storage requirement m. Suppose $|U| = n$ and $m < n/2$, then, if multiple revocation is allowed, $\mathcal{A}$'s re-keying cost is at least $\Theta(n/m)$.*

Proof. First we construct a subset R of U, such that $|R| \leq \frac{1}{2}|U|$ and any set in $\mathcal{A}$ of size $2m$ or above must intersect R. Since $m < n/2$, $2m < n$ and there indeed exist sets of size $2m$. To do this we simply arbitrarily pick one point out of each set of size $2m$ or above. We claim that there cannot be more than $\lceil n/2 \rceil$ sets of size at least $2m$. To prove this claim, we assume the contrary. Since each set covers at least $2m$ users and $|U| = n$, on average, each node is covered more than $\lceil n/2 \rceil \cdot 2m/n$ times. Since $\lceil n/2 \rceil \cdot 2m/n \geq (n/2) \cdot 2m/n = m$, each node is covered more than m time (on average). By the (extended) Pigeonhole Principle, there must be a user u that is covered more than m times, meaning that $Overlap(\mathcal{A}, u) > m$. This contradicts to the assumption $Storage(\mathcal{A}) = m$.

Now we let R be the set of leaving users. Since there cannot be more than $\lceil n/2 \rceil$ sets, and we pick up one point out of each set, $|R| \leq (1/2) \cdot |U|$. To revoke R, GC cannot use any KEK that is shared by more than $2m$ users, because any set in $\mathcal{A}$ of size at least $2m$ must intersect R and therefore must contain at least one leaving node. In other words, GC can only use KEKs shared by at most $2m - 1$ users. Since $|R| \leq (1/2) \cdot |U|$, the number of remaining users that need to be re-keyed (i.e. $|U - R|$) is at least $(1/2) \cdot |U|$. Finally, since there are at least $(1/2) \cdot |U|$ users that need to be re-keyed and each KEK can cover at most $2m - 1$ users, there must be at least $(1/2) \cdot U/(2m - 1)$ re-keying messages, which is of order $\Theta(U/m)$. ■

4.4. Remarks

In the literature, most BE schemes as well as GKD schemes have storage requirement $\log^{\alpha} n$ (e.g. [20,13,35,34,24]). This means BE's re-keying cost is almost as large as $\Theta(n)$ when n is large (Corollary 4.1). In fact, we derived a negative result, as both the lower bounds of GKD and BE show that multiple revocation is a fundamentally time-consuming task. Both of them are far larger than the anticipated log bound $O(\log n)$. In contrast to the log bound of single revocation (such as [35,34,33, 24]), multiple revocation is nearly impractical in light-weight applications (e.g. wireless sensor networks).

Corollary 4.1. *In particular, since most practical broadcast encryption schemes have storage requirement* $\log^{\alpha} n$ *for some* $\alpha > 0$*, their re-keying cost is at least* $\Theta(n/\log^{\alpha} n)$*, which is almost linear.*

5. RBE: Randomized broadcast encryption scheme

Having derived the lower bounds in Sections 3 and 4, we turn our interests to the design of efficient BE schemes. In this section we wil present RBE, our new broadcast encryption Scheme. RBE, a *Randomized Scheme* of *Monte Carlo*[4] fashion, was designed to support efficient mass revocation operations. Its brief descriptions are as follows. GC has l independent KEK key pools, and before the system starts each user draws a KEK from each key pool. To revoke a set of users, GC simply uses the KEKs not held by these users and encrypts the session key update with other KEKs. With a certain probability, a remaining user will have a KEK matching what GC has used to encrypt the session key with. In what follows, we use the language of random functions, which can be thought of as a user drawing a key from the KEK key pool.

5.1. The RBE scheme

5.1.1. Scheme setup

Let N, M be the source and target set of all of our random functions respectively. Let n be the number of nodes in the network. Assume $|N| = n$, $|M| = m$, and $|S| = l$. We construct a family of random functions $\{f_s | f_s : N \to M, s \in S\}$, where S is some index set. (We sometimes denote it by $(f_s)_{s\in S}$ or even (f_s) when the source, target, and index sets are clear from the context.) The set of *KEK*s, namely $\mathcal{K}$, is defined as follows:

$$\mathcal{K} = \{k_{s,t} | s \in S, t \in M\}.$$

Each pair of (s, t) corresponds to a *KEK* and will be generated by GC independently in the initialization stage, so the number of all *KEK*s are lm. Our scheme works as follows.

5.1.2. The scheme

1. *Initialization*: Let $(f_s)_{s\in S}$ be a random function family from N to M with index set S. GC generates $\mathcal{K}$. For each node u_i where $1 \le i \le n$, GC secretly gives u_i the set of his auxiliary keys $\mathcal{K}(u_i) = \{k_{s,f_s(i)} | s \in S\}$ along with the *TEK*.
2. *Re-keying*: If a set of nodes R are compromised by the enemy, GC must re-key *TEK*. To remove R from U where $|R| = r \le m$, GC randomly chooses a key TEK_{new}, encrypts it with every key not belonging to R, separately, and then broadcasts these encrypted messages to all nodes. In short, GC broadcasts
$$\{E_k(TEK_{new}) | k \in \mathcal{K}, k \notin \mathcal{K}(R)\}.$$
3. *Decryption*: Each node $u_i \notin R$ uses one of the *KEK*s $k \in \mathcal{K}(u_i)$ to decrypt $E_k(TEK_{new})$ and obtain TEK_{new}.

The crucial step in our scheme is step 2, since it is not obvious whether we can find such k's (i.e. $k \in \mathcal{K}, k \notin \mathcal{K}(R)$). The following theorem shows that we can find such k's with a really high probability, even if l is small.

Theorem 5.1. *For each user* u_i*, the probability of successfully re-keying this user is close to* $1 - \left(1 - e^{-\frac{r}{m}}\right)^l$*, i.e.*

$$P\left(\mathcal{K}(u_i) \backslash \mathcal{K}(R) \neq \emptyset\right) \approx 1 - \left(1 - e^{-\frac{r}{m}}\right)^l, \quad \forall u_i.$$

Proof. Let us fix u_i. u_i can be re-keyed if and only if $\mathcal{K}(u_i) \not\subseteq \mathcal{K}(R)$. Now we fix s and consider

$$P\left(k_{s,f_s(i)} \notin \{k_{s,f_s(j)} | j \in R\}\right),$$

from which we obtain

$$\begin{aligned} P\left(k_{s,f_s(i)} \notin \{k_{s,f_s(j)} | j \in R\}\right) &= P\left(k_{s,f_s(i)} \neq k_{s,f_s(j)} \forall j \in R\right) \\ &= \left(\frac{m-1}{m}\right)^r = \left[\left(1 - \frac{1}{m}\right)^{-m}\right]^{-\frac{r}{m}} \approx e^{-\frac{r}{m}}. \end{aligned}$$

[4] The term *Monte Carlo* is originally used in randomized algorithms, as opposed to the *Las Vegas* type. Monte Carlo algorithms will always guarantee a correct solution, while Las Vegas will not. With the abuse of language, we use it on our randomized scheme, as they share similar ideas.

Therefore,

$$P\Big(k_{s,f_s(i)} \in \{k_{s,f_s(j)} | j \in R\}\Big) \approx 1 - \mathrm{e}^{-\frac{r}{m}}.$$

Since there are l such s's, we have

$$P\Big(u_i \text{ cannot be re-keyed}\Big) \approx \Big(1 - \mathrm{e}^{-\frac{r}{m}}\Big)^l.$$

Finally, we get

$$P\Big(u_i \text{ can be re-keyed}\Big) \approx 1 - \Big(1 - \mathrm{e}^{-\frac{r}{m}}\Big)^l. \quad \blacksquare$$

Theorem 5.2. *The expected number of keys GC needs to update is approximately* $lm \cdot \mathrm{e}^{-\frac{r}{m}}$.

Proof. This theorem can be proved directly from the linearity of expected values. Let us first define what the *Expected Number of Keys GC needs to update* means. Let Ω be the sample space of the keys held by $R = \{u_1, \ldots, u_r\}$, and for any $\omega \in \Omega$ we define the *characteristic function* $\chi_{i,s}(\omega)$ as follows:

$$\chi_{i,s}(\omega) = \begin{cases} 1, & \text{if } i \notin \omega_s, \text{where } \omega_s = \{f_s(u) | u \in R\} \\ 0, & \text{otherwise.} \end{cases}$$

Define the *Number of Messages* with respect to $s \in S$ as the random variable $X_s(\omega) = \sum_i \chi_{i,s}(\omega)$ and the *Number of Total Messages* as $X = \sum_{s \in S} X_s$. The *Expected Number of Total Messages* is thus the expected value, EX, of X. By the linearity of expected values, we have

$$\begin{aligned} EX &= E\Big(\sum_{s \in S} X_s\Big) = \sum_{s \in S} EX_s = \sum_{s \in S} \sum_{\omega \in \Omega} P(\omega) X_s(\omega) \\ &= \sum_{s \in S} \sum_{\omega \in \Omega} P(\omega) \Big(\sum_i \chi_{i,s}(\omega)\Big) = \sum_{s \in S} \sum_{\omega \in \Omega} \sum_{i=1}^{m} P(\omega) \chi_{i,s}(\omega). \end{aligned}$$

Since $\chi_{i,s}(\omega)$ depends solely on f_s, a completely random function independent of s, it follows that $\chi_{i,s}(\omega)$ does not depend on s. We henceforth use $\chi_i(\omega)$ to represent $\chi_{i,s}(\omega)$, and the outmost summation can be simplified as follows

$$\begin{aligned} \sum_{s \in S} \sum_{\omega \in \Omega} \sum_{i=1}^{m} P(\omega) \chi_{i,s}(\omega) &= \sum_{s \in S} \sum_{\omega \in \Omega} \sum_{i=1}^{m} P(\omega) \chi_i(\omega) \\ &= l \cdot \sum_{\omega \in \Omega} \sum_{i=1}^{m} P(\omega) \chi_i(\omega) = l \cdot \sum_{i=1}^{m} \sum_{\omega \in \Omega} P(\omega) \chi_i(\omega) \\ &= l \cdot \sum_{i=1}^{m} \Big[\sum_{\omega, i \in \omega_s} P(\omega) \chi_i(\omega) + \sum_{\omega, i \notin \omega_s} P(\omega) \chi_i(\omega) \Big] \\ &= l \cdot \sum_{i=1}^{m} \Big[\sum_{\omega, i \in \omega_s} P(\omega) \cdot 0 + \sum_{\omega, i \notin \omega_s} P(\omega) \cdot 1 \Big] \\ &= l \cdot \sum_{i=1}^{m} \Big[0 + \sum_{\omega, i \notin \omega_s} P(\omega) \Big] = l \cdot \sum_{i=1}^{m} \Big(\sum_{\omega, i \notin \omega_s} P(\omega) \Big) \\ &= l \cdot \sum_{i=1}^{m} \Big(P\big(i \notin \omega_s\big) \Big). \end{aligned}$$

Here we view "$i \notin \omega_s$" as an event and use $P\big(i \notin \omega_s\big)$ to denote its probability. The event $i \notin \omega_s$ holds if and only if u_i can be re-keyed by the keys on line s. Therefore, $P\big(i \notin \omega_s\big) = 1 - \Big(1 - \mathrm{e}^{-\frac{r}{m}}\Big)^1 = \mathrm{e}^{-\frac{r}{m}}$, obtained by substitute 1 for l. Plugging it in, we finally get

$$l \cdot \sum_{i=1}^{m} \Big(P\big(i \notin \omega\big) \Big) = lm \cdot \mathrm{e}^{-\frac{r}{m}}.$$

This theorem is thus proved. $\blacksquare$

5.1.3. Storage requirement

In RBE, each user has to save l KEKs, and GC has to save lm KEKs.

Table 1
Storage requirements

	Number of keys	In our example
RBE	l	60
SD	$(1/2)\log^2 n + (1/2)\log n + 1$	147
LSD	$\log^{3/2} n + \log n$	85

5.1.4. Security analysis

We analyze the security of our scheme in the aspects of *Group Key Secrecy, Forward Secrecy, and Backward Secrecy:*

- *Group Key Secrecy*: In our scheme, GC generates TEKs at random and all communications within the group is encrypted with the current TEK. The adversary cannot use any feasible means to discover or compute any TEKs. As a result, the group key secrecy can be guaranteed.
- *Forward Secrecy*: Forward Secrecy can be achieved by revoking the KEKs belonging to the revoked nodes. Once a node gets revoked, all KEKs belonging to him will never be used again. In other words he cannot use his information to decrypt future key updates.
- *Backward Secrecy*: Since this is a broadcast encryption scheme, no join event will happen. Backward secrecy is not in the scope of our discussion.

5.2. Ensuring perfect re-keying

5.2.1. Monte-Carlo-ized RBE

The robustness and performance of the RBE scheme is best described by Theorems 5.1 and 5.2. However, we cannot ensure that the probability of successful re-keying is always 1. In other words, we cannot guarantee that every legal remaining user will get the session key. To remedy this problem, we can let each user share an individual key with GC. Also, just in case that none of the keys can be used to encrypt with , GC can always encrypt the session key with that individual key. Making users sharing individual keys with GC makes our scheme the *Monte Carlo* type, as every legal user is guaranteed to get the session key update and remain in the group.

5.2.2. Improving the performance

We can use the same idea to further reduce the expected message length (expected number of keys GC needs to update). We can start to use individual keys when most users have already been re-keyed. According to Theorem 5.2, the expected number of keys is $lm \cdot e^{-\frac{r}{m}}$. If, instead of l 'lines', only x lines ($x < l$) are used, then the expected number of keys is $xm \cdot e^{-\frac{r}{m}}$. Furthermore, the expected number of remaining users not yet re-keyed will be

$$\left[1 - P(\text{successful re-keying})\right] \cdot (\#\text{ of total remaining users}) = \left[1 - \left(1 - \left(1 - e^{-\frac{r}{m}}\right)^x\right)\right] \cdot (n - r).$$

Since these users will be re-keyed by using individual keys, the expected number of keys GC needs to update will be

$$xm \cdot e^{-\frac{r}{m}} + \left(1 - e^{-\frac{r}{m}}\right)^x \cdot (n - r).$$

Now we view the above expression as a function of x and treat r, m, n as constants. Namely,

$$f(x) = xm \cdot e^{-\frac{r}{m}} + \left(1 - e^{-\frac{r}{m}}\right)^x \cdot (n - r), \quad 0 \le x \le l.$$

Then we can always differentiate $f(x)$ to find its minimum. On the other hand, the value l can be decided based on the minimum of $f(x)$ within $x \in [0, \infty)$.

5.2.3. Simulation results

Fig. 3 shows the performance of a typical example where $n = 10^5$ and $m = 10^4$. The performance is measured as the *length of re-keying message* according to [35] and [20]. In these figures, l ranges from 0 to 100 and r ranges from 0 to 80 000, in which the r-axis is scaled to 1/10 000 (so '8' represents '80 000'). Note that the length of re-keying message is proportional to the number of keys GC needs to update. Fig. 4(a), (b) show slices of Fig. 3 in the cases of $r = 40\,000$ and $l = 60$ respectively. We can use elementary calculus to find the minimum in both cases, as described earlier. Fig. 4(c) shows the performance comparison with SD and basic LSD. We do not compare with the generalized LSD, as it is not practical and of theoretical interest only. In Fig. 4(a), the horizontal axis represents l, while in (b), (c) it represents $10\,000r$. The vertical axis represents the length of re-keying messages in all of them (Fig. 4(a)–(c)). Table 1 shows the storage requirements of RBE, SD, and basic LSD. We also plugged in the values of our example to show the numerical comparison. These values stand for the number of keys a node has to store (on the receiver's end), and they are calculated by plugging in $m = 10^4$, $n = 10^5$, $r = 4{*}10^4$, $l = 60$.

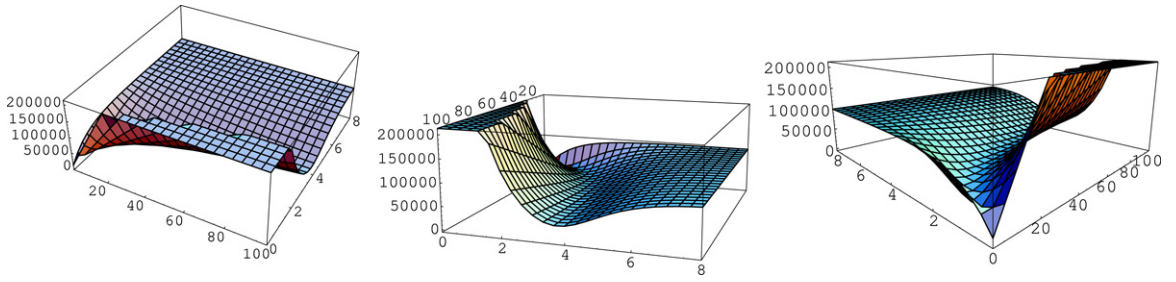

Fig. 3. $m = 10^4$, $n = 10^5$ viewed from 3 different angles.

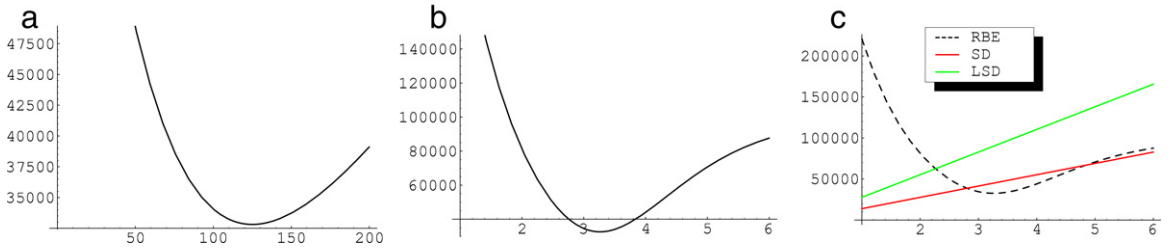

Fig. 4. (a) $r = 40\,000$ (b) $l = 60$ (c) $l = 60$, $m = 10\,000$, $n = 100\,000$.

6. Feasibility of adding join

The lack of *join* often greatly restricts the applicability of broadcast encryption schemes. We want to discuss about the feasibility of doing that in RBE as well as SD, the currently state-of-the-art BE scheme [20]. The twin sister of SD, namely the *Layered Subset Difference* scheme [13], is omitted due to its similar behavior to SD, in this case.

6.0.4. To reuse, or not to reuse?

The fundamental problem about adding join to broadcast encryption schemes is this question. When nodes are revoked, GC never updates anything other than the session key shared by the remaining nodes and the revoked ones in broadcast encryption schemes, as described in Section 2. If these pieces of secret information are not updated, then we cannot reuse these 'slots' once occupied by the revoked users. By simply keeping listening after leaving the group, the revoked users can decrypt some later session key as soon as the secret information is ever used again. Conversely, if these pieces of secret information are updated, then the broadcast encryption Scheme will be turned into a new group key distribution scheme, in which backward secrecy will be ensured by updating the secret information. The goal of this section is to compute how much the updating cost is for the SD and RBE schemes.

6.1. Adding join to SD

We consider the case where one user gets revoked.

Theorem 6.1. *SD scheme's updating cost for a single user's revocation is $O(n \log n)$, where n is the number of users in the group.*

The proof of this theorem is given in the Appendix. According to the theorem, the updating cost for single-user revocation turned out to be prohibitively high, let alone multiple-user revocation. It is simply not practical to do updating in large groups. In the case of LSD, we have similar results as follows.

Theorem 6.2. *LSD scheme's updating cost for a single user's revocation is $O(\sqrt{n} \log n)$, where n is the number of users in the group.*

The proof is omitted, since the computation is almost the same as the case of SD. We list the theorem here for completeness.

6.2. Adding join to RBE

We begin with the following theorem.

Theorem 6.3. *If r users are revoked, the expected number of messages GC has to send to update the secret information of the remaining users is approximately $nl(1 - e^{-\frac{r}{m}})$.*

Proof. Fix a remaining user u_i and an index (a 'line') s. Consider the probability that GC needs to send u_i a replacement key, i.e.

$$\begin{aligned} P\big(\text{GC needs to re-key } u_i\big) &= P\big(\mathcal{K}_s(u_i) \in \mathcal{K}_s(R)\big) \\ &= 1 - P\big(\mathcal{K}_s(u_i) \notin \mathcal{K}_s(R)\big) \\ &= 1 - P\Big(k_{s,f_s(i)} \notin \{k_{s,f_s(j)} | j \in R\}\Big) \\ &= 1 - P\Big(k_{s,f_s(i)} \neq k_{s,f_s(j)}\ \forall j \in R\Big) \\ &= 1 - \Big(\frac{m-1}{m}\Big)^r = 1 - \left[\left(1 - \frac{1}{m}\right)^{-m}\right]^{-\frac{r}{m}} \\ &\approx 1 - \mathrm{e}^{-\frac{r}{m}} \end{aligned}$$

by applying the same techniques as the proof of Theorem 5.2, we can get the expected number of messages GC has to send is approximately $nl(1 - \mathrm{e}^{-\frac{r}{m}})$. ■

Theorem 6.3 tells us that the asymptotic updating cost is $O(n)$ in RBE. Though better than SD, it is still too costly to be practical in large-scale applications. Actually, almost all broadcast encryption schemes have high key-updating cost. The reason is that the key update needs to satisfy the *Key Independence Condition*, otherwise it will suffer from *Reuse Key Attack*. The condition is described as follows:

Definition 6.1 (*Key Independence Condition*). For any user, it is computationally infeasible to derive any key update from others.

The above condition puts a rather strict limitation on GC's key updating methods. Because of this condition, the key updates of the same user cannot be defined as the function of previous keys and the group key. Detailed discussion will be presented later.

6.2.1. Reuse key attack

Suppose the previous key and group key can be used to derive the new update in a broadcast encryption scheme, then the security will be seriously undermined if the invalid keys are used again. Consider the following scenario in a broadcast encryption scheme:

> Let R be the set of revoked users. If a remaining user u shares a key (i.e. $\mathcal{K}(u) \cap R \neq \emptyset$), u just locally replaces each $k \in \mathcal{K}(u) \cap R$ by $h(k, TEK)$, where h is a pre-defined hash function.

The above method is very efficient, as the updating can be done locally by each remaining user. However, it violates the Key Independence Condition, and an attacker can launch Reuse Key Attack as follows:

> Suppose u, v are two colliding users not having exactly the same keys (i.e. there is at least one key different). u leaves the group first while v remains in the group. v then sends u the new group key TEK as it still has membership. u can use TEK to update all of his revoked keys by computing $\{h(k, TEK) | k \in \mathcal{K}(u)\}$. Since u has at least one key, say k_0 different from all keys in $\mathcal{K}(v)$, u will possess a freshly updated key k_0' and use it henceforth, even after v quits the group.

Reuse Key Attack imposed a strong limitation on updating methods in most broadcast encryption schemes. In most cases, the updating cost will be rather costly, and a second-best way to provide join, namely *Overprovisioning*, is more applicable in most cases.

6.3. Overprovisioning

6.3.1. In SD

If the number of users is not an exact power of 2, then the binary tree is not full and there will be some free 'slots' there. In this case, to join a new user, all GC has to do is secretly give that user all necessary information corresponding to one free slot. If the number of users is an exact power of 2, then things will be much more complicated. In this case, GC must "make room" for joining users, and the tree must be expanded. When expansion happens, the size of the tree will be doubled. Let r be the new root, then the old root 0 will become a child of r. Similar to the analysis of Theorem 6.1, GC needs to send the new keys $\{L_{r,t} | t \in T_0\}$ to all remaining users. The following theorem tells us how much the cost is:

Theorem 6.4. *To expand the tree, GC has to send exactly n messages, where n is the number of existing users.*

Proof. We use $L_{a,b}$ to represent the key shared by $S_{a,b}$. GC has to send to any subtree T_x rooted at node x a new key $L_{r,sib(x)}$, where $sib(x)$ is the sibling of x. Since there are n nodes in T_0 and each of which corresponds to a key, GC needs to send n keys in total. ■

Theorem 6.4 tells us that, in order to join a new user, GC needs to send $O(n)$ messages if the tree is full. This rather high cost does not come with surprise: when the binary tree is full and a new node needs to be joined, the cost of join is high due to the cost of restructuring the tree. Fortunately, this will not happen too often, and the *amortized* cost is not too high. In real applications, we normally *overprovision* the keys in order to prevent such tree expansion. The cost of overprovisioning is discussed in the following theorem:

Theorem 6.5. *If n is the number of current users. To overprovision n more users, each user has to store $\lfloor \log n \rfloor + 1$ more keys.*

Proof. To make room for n more users, the tree must be expanded and the depths of all nodes will be increased by 1. Let r be the new root and 0 be the old one. Each node $u \in T_0$ must store the following new keys: $\left\{L_{r,sib(x)} \middle| x \in path(u, r)\right\}$, where $path(u, r)$ is the path from u to r. Since $|path(u, r)| = \lfloor \log n \rfloor + 1$, the theorem is proved. ■

6.3.2. In RBE

Quite unlike SD, RBE has no fixed overhead for overprovisioning. Completely devoid of the idea of 'slots', RBE has much freedom of fine-tuning the performance. Since its performance totally depends on the parameters l, m, n, r, we can set up their values depending on how much trade-off is needed, thus providing tremendous flexibility for practical applications. Generally speaking, the bigger l is, the more 'room' there is for new users joining. l also represents the storage requirement at nodes. If the nodes can store more keys, RBE will have better performance, therefore capable of accommodating more new users.

7. Related work

Broadcast encryption schemes: Fiat and Naor [12], creators of *broadcast encryption* schemes, introduced this idea to handle dynamic membership changes in secure communication groups. Naor et al. [20] later designed an efficient *stateless* broadcast encryption scheme SD, by far the most efficient broadcast encryption schemes in terms of GC's re-keying message size. SD's unsatisfactory users' storage requirement $O(\log^2 n)$ was reduced by Halevy et al. [13], making the scheme more practical to large-scale networks. Halevy et al. also proposed a generalized scheme in [13], allowing more trade-offs. Aside from them, *Perfect Hash Families* were also used to construct different broadcast encryption schemes, such as the work of Safavi-Naini et al. [23] and Fiat-Naor [12].

Group key distribution schemes: Group key distribution protocols are mainly built on top of *logical trees*. This idea was first proposed by Wong et al. [35] and many other protocols followed later, such as [34,33,24]. Although asymptotically they all have the same message overhead $O(\log n)$, McGrew and Sherman's scheme [24], with the smallest constant factor, outperforms others. As a whole, all of these schemes achieve good performance when membership change is relatively not sizable. Canetti et al. [6] proposed a slightly different approach that achieves the same communication overhead. They use the *pseudo-random generator* in their scheme. In these schemes, a new group key must be generated in order to remove a user from the group. These schemes perform well in the case of single join/leave, but they did not handle multiple join/leave efficiently.

Contributory group key distribution schemes: The Diffie–Hellman key exchange protocol [9] is the first important contributory key management protocol, and is also the pioneer of every other protocol of this kind. Later, the following generalizations of D–H protocol to the group scenario came out as well. [15,5,26,28,27] arrange users in a logical ring or chain structure, and accumulate the keying material while traversing group members one by one. [1,3] are also of this kind, in which each node contributes an input to establish a common secret through successive pairwise message exchanges among the nodes in a secure manner using the 2-party D–H exchange [9]. In [32,10,31], logical tree structures are introduced and the number of rounds for establishing the group key is reduced to the logarithm of the group size. Due to their scalability, tree-based schemes are selected as the basic building blocks to address the hierarchical access control problem in distributed environments.

Key pre-distribution schemes: The concept of probabilistic key pre-distribution was proposed by Eschenauer and Gligor in their pioneering work [11]. Chan et al. [8] generalized this idea, by considering q-compositeness, increasing the security and robustness against node loss. Liu and Ning [16], extending the ideas of [11,8], later designed a *polynomial-based framework* by using *random subsets* and *grids*. Zhu et al. [36] designed the LEAP protocol, which uses different methods for different types of transmission. They defined four types of keys: individual, pairwise, cluster, and group keys, and use these keys for different types of transmission. However, no efficient re-keying algorithm is presented in case of node compromise. Inter-node traffic authentication is also mentioned in their paper. Overall, all key pre-distribution schemes are mainly designed for pairwise communications in sensor networks. They primarily focus on problems caused by the limitation of sensor network, such as low computation power, low storage, and unreliable links.

Others: Aside from these three main categories, there are also some other schemes which address different problems. For example, [25,17,4] are mainly focusing on group distribution schemes with self-healing properties. [29] designed a multi-group key management scheme that achieves a hierarchical access control by employing users with various access privileges. Their scheme has low communication, computation, and storage overhead. They also pointed out in [30] the problem of disclosing group membership information caused by key updating information. Bechler et al. [2] used clusters to provide distributed authentication functionalities. Lou et al. [18] proposed the SPREAD scheme to enhance data confidentiality service. Chan [7] presented a distributed key pre-distribution scheme (KDPS) using cryptographic operations. He also

designed an enhanced scheme of Rivest's scheme [22]. Qiang Huang et al. [14] presented two key establishment schemes using symmetric keys. Perrig et al. [21] used exhaustive search in an interesting way to design a pairwise key distribution scheme. Suvo Mittra [19] examined secure unicast and multicast and designed a scalable secure multicasting scheme. They are somewhat related to our work, and we list them here for the sake of completeness.

8. Conclusion and future work

Re-keying in secure group communication is a time-consuming, yet unavoidable task. In this paper, we have established lower bounds for both broadcast encryption and group distribution schemes on the theoretical level, and on the practical level we have designed RBE to meet the lower bounds. However, there are still some interesting questions that remain unsolved using our model, and has the potential to be further explored in the future. For example, in modeling memory requirements, we used the idea of *overlap* to analyze this problem, but for schemes based on pseudorandom generators (such as [20,13]) it cannot be solved. So far, there are still no appropriate mathematical tools to model schemes of this type and we strongly believe that this is an interesting area for us to further delve into.

Appendix

Proof of Theorem 6.1. Again, we use $L_{a,b}$ to represent the key shared by $S_{a,b}$. Suppose the number of users is an exact power of 2. Then we can use a complete binary tree to represent them. If we fix a degree d, then there are exactly 2^d nodes on that level. Let's pick one node j from that level and assume the nodes on the path from the root to j are $\{a_0, a_1, \ldots, a_d\}$, in which a_0 is the root and $a_d = j$. We want to find the common information shared by all nodes of the subtree T_j root at j. Note that if i is an ancestor of j, the common information shared by T_i will be shared by T_j too, as T_j is a subtree of T_i. In our discussion, such information is regarded as T_i's, thus excluded from T_j's. In other words, we want to consider the common information shared *exclusively* by T_j but not by any other nodes. Let the j's sibling be k and consider the keys of $L_{a,b}$ shared by all nodes of T_j. According to Section 2, a, b should satisfy the following condition:

> "a is a common ancestor of all nodes of T_j, and b is the sibling of j".

The common keys shared exclusively by T_j are thus

$$L_{a_0,b}, \quad L_{a_1,b}, \quad L_{a_2,b}, \quad \ldots, \quad L_{a_{d-1},b},$$

in contrast to all keys known by T_j as follows

$$\begin{array}{ccccc} L_{a_0,b_1}, & L_{a_0,b_2}, & L_{a_0,b_3}, & \ldots, & L_{a_0,b_d} \\ & L_{a_1,b_2}, & L_{a_1,b_3}, & \ldots, & L_{a_1,b_d} \\ & & L_{a_2,b_3}, & \ldots, & L_{a_2,b_d} \\ & & & \ddots & \vdots \\ & & & & L_{a_{d-1},b_d}, \end{array}$$

where $b_1, b_2, \ldots, b_{d-1}$ are the siblings of $a_1, a_2, \ldots, a_{d-1}$ and $b_d = b$. From the observation above, we see that the number of keys corresponding to T_j is d, the order of j. On level d, there are 2^d such subtrees. Since GC can multicast a message to a subtree T_j by using a special key S_j shared by all nodes of the subtree (this is a special case of S_{ij} when j is null), the total number of messages GC needs send (measured by the encrypted updating message containing one auxiliary key) is

$$\sum_{d=0}^{\log n} d \cdot 2^d = 2n \log n - \frac{n}{2} + 1 = O(n \log n).$$

This theorem is then proved. ■

References

[1] G. Ateniese, M. Steiner, G. Tsudik, New multiparty authentication services and key agreement protocols, IEEE Journal on Selected Areas in Communications 18 (4) (2000) 628–640.
[2] M. Bechler, H.-J. Hof, D. Kraft, F. Pählke, L. Wolf, A cluster-based security architecture for ad hoc networks, in: INFOCOM, 2004.
[3] K. Becker, U. Wille, Communication complexity of group key distribution, in: 5th ACM Conference on Computer and Communications Security, Nov. 1998.
[4] C. Blundo, P. D'Arco, M. Listo, A new self-healing key distribution scheme, in: IEEE International Symposium on Computers and Communication, 2003.
[5] M. Burmester, Y. Desmedt, A secure and efficient conference key distribution system, in: A. Santis (Ed.), Advances in Cryptology - EUROCRYPT 94, in: Lecture Notes in Computer Science, vol. 950, Springer-Verlag, Berlin Germany, 1995, pp. 275–286.
[6] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, Multicast security: A taxonomy and efficient constructions, in: INFOCOM'99, vol. 2, New York, USA, Mar. 1999, pp. 708–716.
[7] A.C.-F. Chan, Distributed symmetric key management for mobile ad hoc networks, in: INFOCOM, 2004.
[8] H. Chan, A. Perrig, D. Song, Random key predistribution schemes for sensor networks, in: IEEE Symposium on Security and Privacy, May 2003.
[9] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (1976) 644–654.
[10] L.R. Dondeti, S. Mukherjee, A. Samal, DISEC: A distributed framework for scalable secure many-to-many communication, in: IEEE Symposium on Computers and Communications, 2000, pp. 693–698.

[11] L. Eschenauer, V.D. Gligor, A key-management scheme for distributed sensor networks, in: 9th ACM Conference on Computer and Communication Security, Nov. 2002, pp. 41–47.
[12] A. Fiat, M. Naor, Broadcast encryption, in: Advances in Cryptology — CRYPTO'93, 1993.
[13] D. Halevy, A. Shamir, The LSD broadcast encryption scheme, in: Advances in Cryptology CRYPTO'02, 2002, pp. 47–60.
[14] Q. Huang, J. Cukier, H. Kobayashi, B. Liu, J. Zhang, Fast authenticated key establishment protocols for self-organizing sensor networks, in: ACM WSNA, 2003.
[15] I. Ingemarsson, D. Tang, C. Wang, A conference key distribution system, IEEE Transactions on Information Theory 28 (1982) 714–720.
[16] D. Liu, P. Ning, Establishing pairwise keys in distributed sensor networks, in: ACM CCS'03, 2003.
[17] D. Liu, P. Ning, K. Sun, Efficient self-healing group key distribution with revocation capability, in: ACM CCS, 2003.
[18] W. Lou, W. Liu, Y. Fang, SPREAD: Enhancing data confidentiality in mobile ad hoc networks, in: INFOCOM, 2004.
[19] S. Mittra, Iolus: A framework for scalable secure multicasting, in: ACM SIGCOMM'97, Cannes, France, 1997.
[20] D. Naor, M. Naor, J. Lotspiech, Revocation and tracing schemes for stateless receivers, in: Advances in Cryptology- CRYPTO'01, 2001, pp. 41–62.
[21] A. Perrig, D. Song, D. Tygar, Elk, a new protocol for efficient large-group key distribution, in: IEEE Symposium on Security and Privacy, May 2001.
[22] R.L. Rivest, L. Adleman, M.L. Dertouzos, On data banks and privacy homomorphisms, in: Foundations of Secure Computation, 1978, pp. 169–179.
[23] R. Safavi-Naini, H. Wang, New Constructions for Multicast Re-keying Schemes Using Perfect Hash Families, ACM Press, New York, NY, USA, 2000.
[24] A.T. Sherman, D.A. McGrew, Key establishment in large dynamic groups using one-way function trees, IEEE Transactions on Software Engineering 29 (05) (2003) 444–458.
[25] J. Staddon, S. Miner, M. Franklin, Self-healing key distribution with revocation, in: IEEE Symposium on Security and Privacy, 2002.
[26] M. Steiner, G. Tsudik, M. Waidner, Diffie–Hellman key distribution extended to group communication, in: ACM SIGSAC: 3rd ACM Conference on Computer and Communications Security, New Dehli, India, Mar. 1996, pp. 31–37.
[27] M. Steiner, G. Tsudik, M. Waidner, Cliques: A new approach to group key agreement, IEEE Transactions on Parallel and Distributed Systems (August) (2000).
[28] M. Steiner, G. Tsudik, M. Waidner, Key agreement in dynamic peer groups, IEEE Transactions on Parallel and Distributed Systems 11 (8) (2000) 769–780.
[29] Y. Sun, K.J.R. Liu, Scalable hierarchical access control in secure group communications, in: INFOCOM, 2004.
[30] Y. Sun, K.J.R. Liu, Securing dynamic membership information in multicast communications, in: INFOCOM, 2004.
[31] W. Trappe, Y. Wang, K.J.R. Liu, Establishment of conference keys in heterogeneous networks, in: IEEE International Conference on Communications, vol. 4, 2002, pp. 2202–2205.
[32] G. Tsudik, Y. Kim, A. Perrig, Simple and fault-tolerent key agreement for dynamic collaborative groups, in: ACM CCS, 2000.
[33] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, B. Plattner, The VersaKey framework: Versatile group key management, in: Middleware, IEEE Journal on Selected Areas in Communications 17 (9) (1999) 1614–1631 (special issue).
[34] D.M. Wallner, E.J. Harder, R.C. Agee, Key Management for Multicast: Issues and Architectures, 1999.
[35] C.K. Wong, M.G. Gouda, S.S. Lam, Secure group communications using key graphs, IEEE/ACM Transactions on Networking 8 (1) (2000) 16–30.
[36] S. Zhu, S. Setia, S. Jajodia, LEAP: Efficient security mechanisms for large-scale distributed sensor networks, in: ACM CCS'03, 2003.