# Efficient Multicast Key Distribution Using HOWP-based Dynamic Group Access Structures

Jing Liu, Qiong Huang, Bo Yang, and Yang Zhang

**Abstract**— When assigning personal keys, stateful multicast key distribution (MKD) protocols usually rely on some type of dynamic *group access structure* which helps achieve a better tradeoff among storage, communication and computation overheads. However, there exist some stateful MKD protocols whose personal key assignments are based on two static group access structures called *Dual Hash Chain* (DHC) and *Binary Hash Tree* (BHT). We introduce two new types of group access structures called *Dual Homomorphic One-way Function Chain* (D-HOFC) and *Top-Down Homomorphic One-way Function Tree* (TD-HOFT). Both can be regarded as dynamic counterparts of DHC and BHT, respectively. Our research motivation is to investigate what benefits these two new dynamic structures will bring for MKD protocols compared with their static counterparts. Using D-HOFC, we propose a time-based MKD protocol that counters the rejoining member attack on a DHC-based protocol, and a stateful user-based MKD protocol that has a lower computational overhead for Group Controller (GC) than the DHC-based protocol. Using TD-HOFT, we design a stateful user-based MKD protocol that outperforms the original EKT protocol. Performance comparisons and experiment results show that our protocols based on dynamic structures have their own advantages compared with those based on the corresponding static counterparts.

**Index Terms**— multicast key distribution, group access structure, homomorphic one-way permutation (HOWP)

## 1 Introduction

### 1.1 The Problem of Multicast Key Distribution

With rapid evolution of Internet, more and more group-oriented applications have been emerging, for instance, IPTV, DVB (Digital Video Broadcast), videoconferences, interactive group games, collaborative applications, stock quote streaming, and so on. These applications all require a one-to-many or many-to-many group communication mechanism. Providing security services for group communication such as traffic integrity, authentication, and confidentiality usually requires securely establishing a group-shared key called *group key* among privileged group members. This problem is called *group key establishment* in the literature [1]. Compared to its two-party counterpart, secure group key establishment in a dynamic group is more challenging. Group key establishment may be broadly categorized as *group key exchange/agreement* and *group key distribution* (also called *multicast key distribution*). In group key exchange protocols [2],[3], each group member contributes an equal share to the common group key (which is then computed as a function of all members' contributions). In (centralized) multicast key distribution (MKD) protocols, a trust third party called group controller (GC) is responsible for creating a new group key when some change in group membership happens, and securely transferring it to all privileged group members over a broadcast channel. MKD protocols usually aim to solve a more specific problem called *immediate group rekeying*. For security-sensitive commercial applications (e.g. pay-per-view, video-on-demand, and highly classified conferences), each message sent to a group is encrypted with a group key, and the group key must be changed for every membership change. To prevent a new member from decoding messages exchanged before it joins a group, a new group key must be distributed to the group when a new member joins. This security requirement is called *group backward secrecy*. On the other hand, to prevent a departing member from continuing access to the group's communication, the key should be updated as soon as a member leaves. This security requirement is called *group forward secrecy*. To provide both group backward secrecy and group forward secrecy, the group key must be updated (or rekeyed) upon every single change in group membership, and the updated group key must be distributed to all legitimate members. This process is referred to as *immediate group rekeying* in the literature. MKD has been well studied since late 1990s (see [4] for an excellent survey, and more recent surveys are available in [5] and [6]). To the best of our knowledge, tree-based MKD protocols are the most efficient ones to date. Immediate group rekeying following these protocols has $O(\log_2 n)$ communication complexity, and $O(\log_2 n)$ computational and storage complexity for users, where $n$ is group size. The first tree-based MKD protocol is the one based on *Logical Key Hierarchies* (LKH), which was independently

- *J. Liu is with the School of Software, Yunnan University, Kunming 650091 China. E-mail: liujing@ynu.edu.cn.*
- *Q. Huang is with the College of Informatics, South China Agricultural University, Guangzhou 510642, China. E-mail: csqhuang@alumni.cityu.edu.hk.*
- *B. Yang is with the School of Computer Science, Shaanxi Normal University, Xi'an, 710062, China. E-mail: byang@snnu.edu.cn.*
- *Y. Zhang is with the School of Information Science and Technology, Sun Yat-Sen University, Guangzhou, 510006, China. E-mail:llovzy@vip.qq.com.*

suggested by Wong et al. [7], Wallner et al. [8], and Caronni et al. [9]. Since then, a variety of MKD protocols based on LKH [10],[1],[11],[12] have been proposed.

## 1.2 Static vs. Dynamic Group Access Structures

MKD protocols can be subdivided into two categories: the *stateful* and the *stateless*. For stateful MKD protocols, receivers must remain online and keep updating their internal states as long as they still stay attached to the group. If a receiver happens to be off-line when a group rekeying operation occurs, or current rekey message was lost due to a network failure, the receiver will not be able to decipher any future group keys from rekey messages. A successful decipher of current group key depends on successfully receiving all of past rekey messages. Most MKD protocols [7],[8],[9],[10],[1],[11],[12] are stateful. On the contrary, for stateless MKD protocols [13],[14], receivers are not allowed to maintain any internal state. Personal keys are given to registered receivers or reserved for prospective receivers in a setup phase and remain unchanged thereafter. Statelessness property is very desirable for application scenarios where no feedback channel exists (e.g., encrypted DVD distribution) or group members go off-line frequently or communication channel is lossy. Typical stateless MKD protocols are those based on the *subset cover framework* [13]: the complete sub-tree (CS) protocol [13], the subset difference (SD) protocol [13], and the layered subset difference (LSD) protocol [14].

When assigning personal keys to group members, efficient stateless (resp. stateful) MKD protocols usually rely on some type of static (resp. dynamic) cryptographic structures such as key chains or logic key trees, which helps achieve an ideal tradeoff among storage, communication and computational overheads. We call these cryptographic structures *group access structures* to distinguish them from traditional *access structures* (e.g., secret sharing) [15]. A stateful (resp. stateless) user-based MKD protocol can be regarded as consisting of a personal key assignment algorithm based on certain type of dynamic (resp. static) group access structures, and join/leave rekeying algorithms (resp. broadcast encryption algorithm) that are again based on the personal key assignment algorithm. Therefore, group access structures are fundamental to MKD protocols. This fresh view centered on group access structures helps us gain an insight into modular design of MKD protocols.

For most stateless MKD protocols, assignment of personal keys to registered and prospective users in the setup phase is usually based on some kind of pre-specified group access structures. Typical group access structures used by stateless MKD protocols are *complete sub-tree* [13], *subset difference* [13],[14], *flat table* [16],[17], *polynomial interpolation* [18],[19],[20], *Chinese remainder theorem* [21], *dual hash chain* [22],[23] and *top-down one-way function tree* [13],[24]. These group access structures must be pre-specified in the setup phase and then remain unchanged irrespective of group dynamics. That is why we say they are static. Shares or nodes (i.e., personal keys) associated with these group access structures cannot be reassigned to other users even if their holder has left the group, otherwise group backward secrecy would be violated. Therefore, these static group access structures must be big enough to accommodate all future/prospective users at the very beginning. For large and dynamic groups, that may cause huge storage and computational requirements both for GC and end users.

For stateful MKD protocols, assignment of personal keys to a joining user is on-the-fly, and usually based on some kind of dynamically-changing group access structures. Typical group access structures used by stateful MKD protocols are *logic key hierarchies* [7],[8],[9],[10],[11],[12], *bottom-up one-way function tree* [1],[25], *flat table* [12],[26],[27], *dual hash chain* [28],[23] and *top-down one-way function tree* [29],[23]. When a user joins the group, GC needs to first create one or several new nodes/shares on these group access structures for it, and then update relevant keys before assigning them to the joining member to ensure group backward secrecy. When a user leaves the group, GC needs to delete its associated node from these group access structures, and then update those keys held by the evictee (thus all information about the group access structure held by the evictee is invalidated) to ensure group forward secrecy. In a word, these group access structures keep expanding, contracting, and changing as members join or leave. Their current size exactly corresponds to the current number of group members. That is why we say they are *dynamic*. Therefore, most stateful MKD protocols based on dynamic group access structures are more suitable for immediate rekeying for large and dynamic groups than those stateless MKD protocols based on static group access structures.

## 1.3 User-Based MKD Protocols and Time-based Ones

Access to group keys can be controlled through two methods — the *user-based* one [7],[8],[9],[1],[11],[12] and the *time-based* one [30],[24]. The former is more intuitive and traditional. A current group key should be only accessible to current legitimate members, which is achieved by performing group rekeying upon every change in group membership. On the other hand, if every user's departure time can be predetermined at the time of join, GC can divide the group's lifetime into time slots and for every time slot, generate a unique group key that is used to encrypt application data transmitted during that period. When a new member joins the group, it will be provided with some intermediate seeds that can be used to derive those group keys corresponding to the predetermined duration over which it will stay attached to the group. In contrast to traditional user-based MKD protocols, group rekeying following this way is *automatic* (i.e., irrespective of membership dynamics), stateless, and requires transmitting no rekey message. These merits are collectively referred to as *zero side-effect* by Briscoe [24]. We also call this type of group rekeying *automatic group rekeying*.

## 1.4 Research Motivation and Contributions

For the first time, we introduced the concept of group access structure for MKD protocols and provided a fresh view on modular design of MKD protocols which is centered on group access structures. This view helps gain an insight into MKD protocol design. We also revealed the subtle differences between dynamic group access structures and static group access structures.

Some of existing stateless and stateful MKD protocols are based on two useful types of static group access structures — *Dual Hash Chain* (DHC) [24],[28] and *Binary Hash Tree* (BHT) [31],[24]. In this paper, we introduced two new types of group access structures based on *Homomorphic One-Way Functions* (HOWFs): *Dual Homomorphic One-way Function Chain* (D-HOFC) and *Top-Down Homomorphic One-way Function Tree* (TD-HOFT), both of which can be regarded as the dynamic counterparts of DHC and BHT, respectively. We introduced two structure-preserving operations — *chain product* and *tree product*— for updating these structures without compromising their structures. For a particular type of TD-HOFT called *exclusive key tree* (EKT), we introduced an operation called *tree blinding*.

In this paper, we want to investigate what benefits these two new types of dynamic structures — D-HOFC and TD-HOFT — will bring for MKD protocols based on them compared with their static counterparts (DHC and BHT). Our following research results show that these structures have their own advantages over their static counterpart in designing efficient and secure MKD protocols: (1) Employing D-HOFC, we are able to propose a stateful time-based MKD protocol that counters the rejoining member attack on a DHC-based protocol, and a stateful user-based MKD protocol has a lower computational overhead for Group Controller (GC) than the corresponding DHC-based protocol in 60% time of a MBone audio session; (2) Taking full advantage of TD-HOFT, we designed a stateful user-based MKD protocol called EKT+ which outperforms the original EKT protocol; (3) We made both quantitative and qualitative comparisons between our proposed protocols based on dynamic group access structures and those based on the corresponding static counterparts. To further investigate how performance changes with time in practice for every related user-based MKD protocol, we also performed an experiment using MBone user activity data. Both the comparisons and experimental results show that: (a) protocols based on tree-based group access structures have a better performance than those based on chain-based structures; (b) compared with the protocol based on BHT, protocols based on the corresponding dynamic counterpart — TD-HOFT — are more collusion-resistant and have a scalable storage overhead while maintaining computational efficiency comparable to the former; (4) Dynamic structures, D-HOFC and TD-HOFT, are more suitable for designing a hybrid stateful MKD protocol with collusion-bandwidth tradeoffs than their static counterparts, DHC and BHT. In addition, we extended a recently-proposed symbolic security model for user-based MKD protocols to support time-based MKD protocols. We presented rigorous security proofs for all proposed protocols in a symbolic security model.

The rest of this paper is organized as follows. In Section 2, we introduce two general types of group access structures, D-OFC and TD-OFT, and review some of existing MKD protocols based on certain instantiations of them. Section 3 introduces two new types of group access structures — D-HOFC and TD-HOFT, and related operations on them. In Section 4, employing dynamic D-HOFC, we propose both a stateful time-based MKD protocol and a stateful user-based MKD protocol. Taking full advantage of TD-HOFT, we propose a stateful user-based MKD protocol called EKT+ that outperforms the original EKT protocol. Section 5 gives rigorous security proofs for our protocols in a symbolic security model. In Section 6, we give comprehensive comparisons between our protocols based on dynamic group access structures, and those based on corresponding static counterparts. We also perform an experiment to show how performance changes with time in practice for every related user-based MKD protocol. Section 7 concludes this paper and gives some interesting topics for future research.

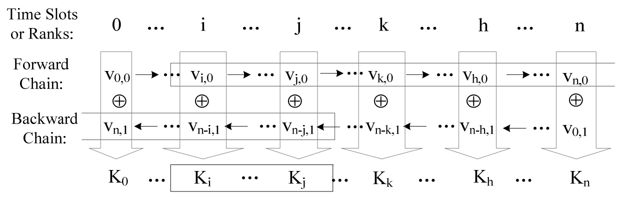
## 2 Two Types of Group Access Structures and 1-Resilient MKD Protocols Based on Them

We first introduce a concept called *1-resilient* that will be used throughout the rest of this paper. According to the concept of *k-resilience* given by Fiat and Naor [31], a protocol is called *1-resilient* if it is secure against any single-user attack, but a coalition of two users might break its security.

In the rest of this paper, we denote by $U$ the universe of all users by $U = \{1, 2,\ldots, n\}$. Denote the privileged subset of users at time $t$ by $S^{(t)} \subset U$, and the complement of set $S^{(t)}$ with respect to $U$ by $R^{(t)}$. For an arbitrary set $S$, we use $|S|$ to denote the order of $S$. The group key used to encrypt those data sent to $S^{(t)}$ is denoted by $GK^{(t)}$. We use $\{M\}_K$ to denote the encryption of $M$ by $K$. In addition, we denote "multicast" by "$\Rightarrow$"and "secure unicast" by "$\rightarrow$".

In the following, we first introduce two general types of group access structures, respectively called *Dual One-way Function Chain* (D-OFC) and *Top-down One-way Function Tree* (TD-OFT). Then we review those 1-resilient MKD protocols based on certain instantiations of them.

### 2.1 D-OFC


Time Slots or Ranks:
0 … i … j … k … h … n
Forward Chain:
$v_{0,0}$ → … $v_{i,0}$ → … $v_{j,0}$ → … $v_{k,0}$ → … $v_{h,0}$ → … $v_{n,0}$
⊕
Backward Chain:
$v_{n,1}$ ← … $v_{n-i,1}$ ← … $v_{n-j,1}$ ← … $v_{n-k,1}$ ← … $v_{n-h,1}$ ← … $v_{0,1}$
$K_0$ … $K_i$ … $K_j$ … $K_k$ … $K_h$ … $K_n$
Fig. 1 Time-based MKD using D-OFC (or DHC)

Referring to Figure 1, *Dual One-way Function Chain* (D-OFC) is composed of two one-way function chains: a forward chain and a backward chain. Both chains are derived respectively from two different initial seeds, say $v_{0,0}$ and $v_{0,1}$, by repeatedly applying a one-way function $h$. That is, the $(i+1)$-th intermediate seed $v_{i+1,B}$ is computed as $v_{i+1,B}=h(v_{i,B})$ ($i$=0,…,$n$; $B$=0, 1). Note that specific candidates for this one-way function $h$ in practice can be a one-way hash function, a homomorphic one-way function (HOWF) or any other particular type of one-way function. Each intermediate seed on a forward/backward chain can be used to compute a group key $K_i$ in a time-based MKD protocol as illustrated in Figure 1 or be assigned to a user as its personal key in a user-based MKD protocol (see Figure 2).

### 2.1.1 A time-based MKD protocol

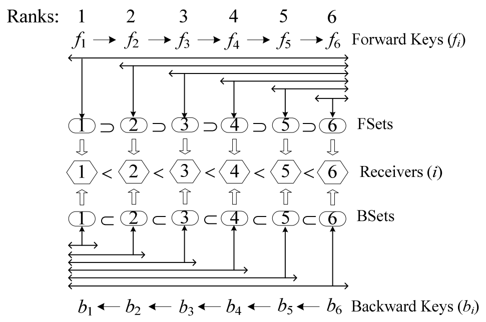
Briscoe [24] proposed a structure called *Bi-directional Hash Chain* that can be regarded as a hash-based instantiation of D-OFC since it is derived by using a one-way hash function in the same manner as above. In the rest of this paper, we would rather call it *Dual Hash Chain* (DHC) because we found the meaning of the original term is somewhat self-contradictory (a hash chain must be one-way). In [24], Briscoe also proposed a time-based MKD protocol based on DHC which allows different portions of a key sequence to be reconstructed from combinations of two intermediate seeds. For convenience, we call it the DHC protocol. We denote the group key used to encrypt those data transmitted during the $i$-th time slot by $K_i$. As illustrated in Figure 1, if we want to restrict a joining member $u_i$ to the data transmitted from time slot $i$ up to time slot $j$ which it has paid for, GC only needs to supply it with two intermediate seeds $v_{i,0}$ and $v_{n-j,1}$ (called *control pair*) when it joins the group. With this pair, $i$ can derive the contiguous key sequence from $K_i$ to $K_j$ by itself. For convenience, we denote the time slot interval from $i$ to $j$ by $[i, j]$, and call it the *authorized time slot interval* for $u_i$.

However, any member cannot be granted access to multiple disjoint sub-sequences of the same key sequence, otherwise this protocol is not secure even in the presence of a single user (i.e., not 1-resilient). Suppose that we want to grant a member $u_i$ two disjoint authorized time slot intervals $[i, j]$ and $[k, h]$ ($k>j$), it will be supplied with two control pairs $\{v_{i,0},v_{n-j,1}\}$ and $\{v_{k,0},v_{n-h,1}\}$ according to the protocol. With these control pairs, the longest possible key sequence that $u_i$ is able to reconstruct is the one from $K_i$ straight to $K_h$, which however contains a sub-sequence from $K_{j+1}$ to $K_{k-1}$ unauthorized for $u_i$. For the same reason, a previously evicted member is disallowed to rejoin a group. For convenience, we call this type of attack *rejoining member attack*. This attack renders the DHC protocol useless in practice. Also for a similar reason, collusion between an arbitrary pair of users would also break security of the DHC protocol. Therefore, the DHC protocol is 1-resilient.

Another drawback with this protocol is that the lifetime of a group must be pre-specified in the setup phase, and it cannot be extended thereafter even when all time slots are used up. Because the backward hash chain of a DHC cannot be extended in the reverse direction.

### 2.1.2 User-based MKD protocols

Fan et al. [28] proposed a 1-resilient user-based stateful MKD protocol called *Linear Ordering of REceivers* (LORE). We first introduce its personal key assignment algorithm that is based on DHC. Let $N$ denote the total number of prospective receivers. Suppose that the entire receivers



Ranks: 1 2 3 4 5 6
$f_1$ $f_2$ $f_3$ $f_4$ $f_5$ $f_6$ Forward Keys ($f_i$)
1 ⊃ 2 ⊃ 3 ⊃ 4 ⊃ 5 ⊃ 6 FSets
1 < 2 < 3 < 4 < 5 < 6 Receivers ($i$)
1 ⊂ 2 ⊂ 3 ⊂ 4 ⊂ 5 ⊂ 6 BSets
$b_1$ $b_2$ $b_3$ $b_4$ $b_5$ $b_6$ Backward Keys ($b_i$)

Fig. 2 Assignment of control keys in LORE

are already linearly ordered as in Figure 2. Each receiver $i$ holds a set of forward keys, denoted by FSet($i$), and a set of backward keys, denoted by BSet($i$). Both sets of keys are called *control keys*. According to the key assignment algorithm as illustrated in Figure 2, for receiver $i$, we have FSet($i$) = $\{f_k \mid i \le k \le N\}$ and BSet($i$) = $\{b_k \mid 1\le k \le i\}$. The effect of such control key assignment is that for any forward key $f_i$, it is known only to receivers with rank no more than $i$, and for any backward key $b_i$, it is known only to receivers with rank no less than $i$. However, this kind of key assignment algorithm requires GC to generate and manage a huge number of keys. One simple solution to this problem is to derive all forward keys (resp. backward keys) from a single seed $f_1$ (resp. $b_N$) by repeatedly applying a one-way hash function like DHC.

Now we describe its group rekeying algorithms. When receiver $i$ joins the group at time $t$, GC sends the following rekey message: GC→ $i$: $GK^{(t+1)}$, $\{f_i, b_i\}$. With control pair $\{f_i, b_i\}$, receiver $i$ can derive all its control keys. For the remaining receivers, GC simply broadcasts the following rekey message:

$$\text{GC}\Rightarrow S^{(t+1)}\text{: } i, \left\{GK^{(t+1)}\right\}_{GK^{(t)}} .$$

All receivers except $i$ can extract the new group key $GK^{(t+1)}$ from this message. When receiver $i$ leaves the group at time $t$, GC multicasts the following rekey message:

$$\text{GC}\Rightarrow S^{(t+1)}\text{: } i, \left\{\left\{GK^{(t+1)}\right\}_{f_{i-1}}\right\}_{GK^{(t)}} \ (if\ i < N),$$

$$\left\{\left\{GK^{(t+1)}\right\}_{b_{i+1}}\right\}_{GK^{(t)}} \ (if\ i > 1)\text{, where } S^{(t+1)}=S^{(t)}\text{-}i.$$

According to the assignment of control keys, all current members except $i$ can extract the new group key $GK^{(t+1)}$ from this message after double decryption.

A coalition of an arbitrary pair of members can break group forward secrecy of the LORE protocol. Suppose that members $i$ (with rank $i$) and $j$ (with rank $j>i$) are both member of $S^{(t)}$. According to personal key assignment algorithm, $i$'s (resp. $j$'s) control pair is $(f_i,b_i)$ (resp. $(f_j, b_j)$). Colluding with $j$, member $i$ can exchange its forward key $f_i$ for $j$'s backward key $b_j$. Thus, when member $i$ (resp. $j$) leaves the group at a later time $t'$ ($t'>t$), it can use $b_j$ (resp. $f_i$) to derive $b_{i+1}$ (resp. $f_{j-1}$), and then obtain the new group key $GK^{(t'+1)}$ by double decrypting the second (resp. first) part of the rekey message transmitted by GC at time $t'$. For the same reason, a rejoining member is not allowed to be assigned a new rank. In fact, this kind of collusion attack is inherent with the D-OFC based personal key assignment algorithm. For convenience, we call this type of collusion attack *member collusion attack*.

In addition, the LORE protocol is also vulnerable to another type of collusion attack called *non-member collusion attack*. We analyze this type of attack through the following two cases:

1) *Coalition between an arbitrary pair of evictees* — Suppose that user $i$ (with rank $i$) left the group at the current time $t$ and user $j$ (with rank $j<i$) had left before time $t$. We denote by $M^{(t)}$ the rekey message transmitted by GC when $i$ left at time $t$. Colluding with $j$, user $i$ can first decrypt the outer encryption of the first part of $M^{(t)}$ (i.e., $\left\{\left\{GK^{(t+1)}\right\}_{f_{i-1}}\right\}_{GK^{(t)}}$) to get $\left\{GK^{(t+1)}\right\}_{f_{i-1}}$, and then transfer it to user $j$ who can decrypt it to get $GK^{(t+1)}$ (note that user $j$ can derive $f_{i-1}$ from its forward control key $f_j$ since $i>j$). Thus, the group forward secrecy is broken;
2) *Coalition between a former evictee and a later-joining member* — Suppose that user $i$ left the group at time $t$ and user $j$ joined at time $t$+1. Using a similar argument as above, it is readily seen that collusion between user $i$ and user $j$ is also successful. In this case both group forward secrecy (with regard to the evictee $i$) and group backward secrecy (with regard to the joining member $j$) are broken.

In either case, both $i$ and $j$ are non-members with respect to the privileged user set $S^{(t+1)}$. Therefore, we call this type of collusion attack *non-member collusion attack*. In a word, collusion between an arbitrary pair of non-members will break the security of the LORE protocol.

## 2.2 TD-OFT

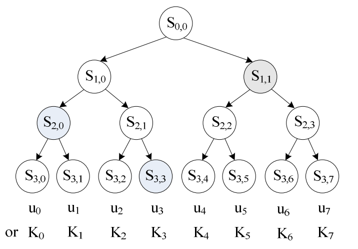


Fig. 3 TD-OFT (or BHT)

ıs illustrated in Figure , a *Top-Down One-way* *unction Tree* (TD-OFT) ; a balance binary tree /hich is derived from single root seed $S_{0,0}$ in top-down manner uch that $S_{i+1,2j}= f_L(S_{i,j})$, $_{i+1,2j+1}= f_R(S_{i,j})$ ($i$=0,1,…) /here $f_L$ and $f_R$ denote wo distinct one-way functions respectively, and $S_{i+1,2j}$ and $S_{i+1,2j+1}$ the left child and the right child of a seed $S_{i,j}$ respectively. Each leaf node $S_{\log n,i}$ ($n$ is the group size) of a TD-OFT will be associated with a user $u_i$ in a user-based MKD protocol, or represent a group key corresponding to a time slot in a time-based MKD protocol.

Briscoe [24] proposed a top-down binary tree structure called *Binary Hash Tree* (BHT) which can be regarded as a hash-based instantiation of TD-OFT since it is derived by using one-way hash functions in the same manner as TD-OFT. Employing BHT, Briscoe [24] proposed a 1-resilient time-based MKD protocol that overcomes some drawbacks with the DHC protocol.

Liu and Wang [23] proposed a 1-resilient stateful user-based MKD protocol based on BHT. For convenience, we call their protocol the LW protocol. The LW protocol is based on a BHT-based personal key assignment algorithm that is similar to the one originally suggested by Fiat and Naor [31]. This algorithm assigns personal keys or seeds to users simply as follows. As illustrated in Figure 3, when a user $u_i$ joins the group, GC associates it with a free leaf node, say $S_{\log n,i}$ on the BHT tree, and supplies it with all seeds associated with the siblings of the nodes on its path to the root. Given these seeds, $u_i$ can compute all the seeds except those on its path to the root. It is readily seen that according to this algorithm, $S_{\log n,i}$ is shared by all users except $u_i$, and thus can be used to exclude $u_i$ from the group. It is based on this sense that $S_{\log n,i}$ is called an *exclusive key* for $u_i$ by Kim et al. [29]. For example, user $u_2$ associated with $S_{3,2}$ is supplied with $S_{3,3}$, $S_{2,0}$, and $S_{1,1}$. And $S_{3,2}$ is an exclusive key for $u_2$ that is shared by all users except $u_2$. When a user $u_i$ leaves the group, GC multicasts the following rekey message:

GC⇒$S^{(t+1)}$: $u_i, \left\{GK^{(t+1)}\right\}_{S_{\log n,i} \oplus GK^{(t)}}$ , where '⊕' represents the exclusive-or operation and $S^{(t+1)}=S^{(t)}-u_i$.

According to the personal key assignment algorithm, all current members except $u_i$ can extract the new group key $GK^{(t+1)}$ from this message. Using a similar argument as given in the end of Section 2.1.2, it is readily seen that the LW protocol is also vulnerable to both member collusion attack and non-member collusion attack as the LORE protocol.

Kim et al. [29] proposed a 1-resilient stateful user-based MKD protocol with similar personal key assignment as that of the LW protocol except that instead of using one-way hash functions, they chose to use HOWFs to facilitate updating the whole TD-OFT when a member leaves or joins the group. For convenience, we call their protocol the *Exclusive Key Tree* (EKT) protocol since it is based on an important idea called *exclusive key* [29],[23]. In contrast to BHT used by the LW protocol, TD-OFT used by the EKT protocol is dynamic. The leave rekeying algorithm of the EKT protocol is similar to that of the EKT+ protocol (refer to Section 4.3). Its join rekeying algorithm is similar to that of the LORE protocol except that the whole TD-OFT must be updated, and a root incremental seed used to update TD-OFT is also encrypted in a rekey message besides the new group key. The EKT protocol suffers similar member collusion attack as the LORE and LW protocols, but a particular type of non-member collusion attack is prevented. Since both the EKT and EKT+ protocols suffer the same collusion attack, we defer further discussion to Section 4.3.

Because both DHC and BHT are derived in either a chain manner or a top-down tree manner by using one-way hash functions, we cannot update any node of them without replacing the whole chain or tree with a new one. In this sense, we can regard DHC as a static instantiation of D-OFC, and BHT as a static instantiation of TD-OFT.

## 3 D-HOFC AND TD-HOFT

Recall that a *one-way function* is a function that is easy to compute on every input, but hard to invert given the image of a random input. In this section, we instantiate D-OFC and TD-OFT by using HOWFs to obtain two new types of group access structures, respectively named *dual homomorphic one-way function chain* (D-HOFC) and *top-down homomorphic one-way function tree* (TD-HOFT). Updating D-HOFC (resp. TD-HOFT) can be easily achieved by performing a *chain product* (resp. a *tree product*) of the original structure and a corresponding incremental structure. Before we give their formal definitions, let's review some basic mathematical concepts about *homomorphism*. We use $(G, *)$ to denote a group $G$ with its algebraic operation "$*$". Given two groups $(G, *)$ and $(H, \cdot)$, a *group homomorphism* from $(G, *)$ to $(H, \cdot)$ is a function $f: G \rightarrow H$ such that for all $u$ and $v$ in $G$, it holds that $f(u*v) = f(u)\cdot f(v)$. A *self-homomorphism* is a group homomorphism that maps a group $G$ to itself. If a *self-homomorphism* is a one-to-one mapping, we call it *homomorphic one-way permutation* (HOWP). If every node of a structure is an element of a group $G$, we say this structure is *defined over G*.

### 3.1 HOFC and D-HOFC

***Definition 1 HOFC —*** An HOFC of length $N$ defined over a group $(G, *)$ and an HOWP $f$ is a one-way chain that is computed by repeatedly applying $f$ in a forward manner as follows. For an arbitrary node $x_i$ in an HOFC $X$, its succeeding node $x_{i+1} = f(x_i)$ $(i = 0,\cdots, N\text{-}2)$.

***Definition 2 Chain product —*** Given two arbitrary HOFCs of the same length, $X$ and $Y$, defined over a group $(G, *)$ and a HOWP $f$, a chain product of $X$ and $Y$, denoted by $X*Y$, is computed by multiplying their corresponding nodes.

**Theorem 1:** *Given two arbitrary HOFCs of the same length, X and Y, both defined over a group (G, *) and an HOWP f, the result of a chain product X * Y is also an HOFC.*

Proof: Let $Z$ be the result of a chain product of $X$ and $Y$, i.e., $Z = X*Y$. We prove for an arbitrary $i$ $(0\leq i\leq N\text{-}1)$, $z_{i+1} = f(z_i)$. Then the theorem would follow immediately according to Definition 1. In fact, we have $z_{i+1}= x_{i+1}*y_{i+1}=f(x_i) *f(y_i)=f(x_i*y_i)=f(z_i)$. □

***Definition 3 D-HOFC —*** D-HOFC of length $N$, defined over a group $(G, *)$ and an HOWP $f$, consist of a forward HOFC and a backward HOFC, both of length $N$ and defined over the group $(G, *)$ as well as the HOWP $f$.

### 3.2 TD-HOFT

***Definition 4 TD-HOFT —*** A TD-HOFT defined over a group $(G, *)$ and two HOWPs $f_L$ and $f_R$ is a balanced binary tree that is derived in the following top-down manner: for an arbitrary node $x_i$ in an HOFT $X$, suppose its left child and right child are denoted by $x_{2i}$ and $x_{2i+1}$ respectively, then we have $x_{2i} = f_L(x_i)$ and $x_{2i+1} = f_R(x_i)$.

To be used as an access control structure, TD-HOFT must at least satisfy the following two conditions: (1) its leaf nodes must be *collision-free*; (2) its leaf nodes must be *independent* (from an arbitrary set of leaf nodes, it is *computationally infeasible* to compute any leaf node outside this set).

***Definition 5 Tree product —*** Given two arbitrary TD-HOFTs of the same depth, $X$ and $Y$, both defined over a $(G, *)$ and two HOWPs $f_L$ and $f_R$, a tree product of $X$ and $Y$, denoted by $X*Y$, is computed by multiplying their corresponding nodes.

**Theorem 2:** *Given two arbitrary TD-HOFTs X and Y with the same depth, both defined over a group (G, *) and two HOWPs $f_L$ and $f_R$, the result of a tree product X * Y is also a TD-HOFT.*

Proof: Let $Z = X*Y$. For an arbitrary node secret $z_i \in Z$, we have $z_{2i}=x_{2i}*y_{2i}=f_L(x_i)*f_L(y_i)= f_L(x_i*y_i)= f_L(z_i)$. For the same reason, we have $z_{2i+1} = f_R(z_i)$. Thus, $Z$ is a TD-HOFT according to Definition 4. □

***Definition 6 Tree blinding —*** Given an arbitrary TD-HOFT $X$, a tree blinding of $X$ maps $X$ to another key tree $Y$, denoted by $Y=B(X)$ such that (1) $Y$ is still a TD-HOFT, (2) from any set of nodes of $Y$, it is computationally infeasible to compute any node of $X$.

For an arbitrary type of HOWP-based TD-HOFT, a tree blinding operation may or may not exist. Theorem 3 (refer to Section 4.3.1) shows that it does exist for a particular type of TD-HOFT — the Blum-Williams function based one.

Theorem 1 and Theorem 2 show that both chain product and tree product are *structure-preserving* operations. A tree blinding operation on a TD-HOFT helps conceal information about its every node without using any additional incremental structure and compromising its structure.

### 3.3 Efficient Candidates for HOWP

Considering computational efficiency, *Rabin function* (modular squaring) [32] and *RSA function* [33] with small encryption exponent (e.g., 3) are both good candidates for homomorphic one-way functions. Rabin function is more computationally-efficient than RSA function. However modular squaring is not a permutation on $Z_n^*$, it is in fact a 4-to-1 mapping. But a variant of Rabin function — the *Blum-Williams function* is a trapdoor permutation on $QR(n)$ where $QR(n)$ denotes the set of all quadratic residues modulo $n$. Blum-Williams function is defined as follows: for an integer $x \in Z_n^*$, compute $y = x^2 \bmod n$, where $n$ is a *Blum integer* (i.e., $n$ is a product of two distinct primes each congruent to 3 modulo 4). In a word, the Blum-Williams function is the best candidate for HOWP.

## 4 MKD PROTOCOLS BASED ON D-HOFC AND TD-HOFT

As discussed in Section 1.2, when assigning personal keys to new members, efficient stateful MKD protocols usually rely on some type of dynamic group access structure which helps achieve a better tradeoff among storage, communication and computation overheads. However, we have already seen in Section 2 that some stateful MKD protocols use two static types of group access structures (DHC and BHT) instead. In this section, we further investigate what benefits the two new dynamic group access structures (D-HOFC and TD-HOFT) will bring for MKD protocols based on them compared with their static counterparts. We use D-HOFC to design both a stateful time-based MKD protocol and a stateful user-based MKD protocol. The former counters the rejoining member attack on the DHC-based protocol. One of our experiment results given in Section 6.4 shows that the latter has a lower computational overhead for GC than the corresponding DHC-based protocol in 60% time of a MBone audio session. We also take full advantage of TD-HOFT to design a stateful user-based MKD protocol called EKT+ that outperforms the original EKT protocol. In the end of this section, we reveal that 1-resilient stateful MKD protocols based on dynamic group access structures are more suitable for constructing a hybrid stateful MKD protocol with collusion-bandwidth tradeoffs than those based on static group access structures.

## 4.1 The Stateful Time-Based MKD Protocol — Protocol I

To counter the rejoining member attack on the DHC protocol that renders it useless, the idea is that whenever detecting a former evictee $u_i$ is rejoining the group at time slot $c$, GC updates both the forward HOFC and the backward HOFC (excluding all those nodes on both chains belonging to the past time slot interval [0, $c$-1]) by multiplying them by a corresponding incremental HOFC before supplying $u_i$ with its new control pair. To allow other legitimate members to update their control pairs, GC only needs to broadcast two incremental root seeds. In this way, GC establishes a whole new D-HOFC among the group without rekeying the entire group (i.e., sending each member its new control pair). Since the two control pairs held by the rejoining member now belong to different D-HOFC, it cannot use them to derive any unauthorized group key. Thus, the rejoining member attack is prevented. However, the same method also causes a drawback that GC must record a long history revocation list during the group's lifetime for detecting a rejoining member.

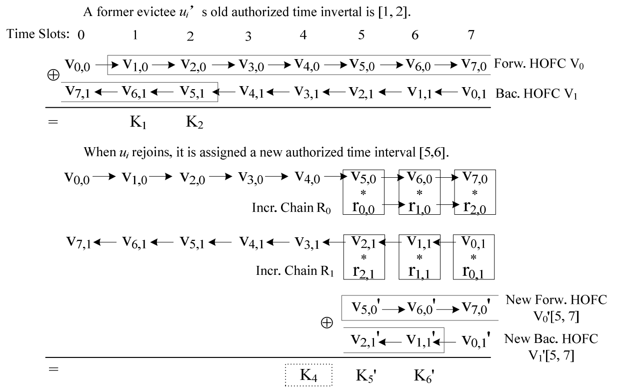

A former evictee $u_i$' s old authorized time invertal is [1, 2].
Time Slots: 0 1 2 3 4 5 6 7
Forw. HOFC $V_0$
Bac. HOFC $V_1$
$K_1$ $K_2$
When $u_i$ rejoins, it is assigned a new authorized time interval [5,6].
Incr. Chain $R_0$
Incr. Chain $R_1$
New Forw. HOFC $V_0'[5, 7]$
New Bac. HOFC $V_1'[5, 7]$
$K_4$ $K_5'$ $K_6'$

Fig. 4 A time-based MKD with 8 time-slots

### 4.1.1 Rejoin rekeying

Since the following group rekeying operation is performed by GC when a former evictee rejoins the group, we call our algorithm *rejoin rekeying*. As illustrated in Figure 4, suppose that the group's lifetime is $l$ time slots, and the corresponding D-HOFC consists of a forward HOFC $V_0$ and a backward HOFC $V_1$. When a former evictee $u_i$ rejoins the group (suppose that its old authorized time slot interval is [$a$, $b$]), GC detects it is a rejoining member and assigns it a new authorized time slot interval [$c$, $d$]. After that, GC performs the following steps to complete rejoining rekeying: (1) ***Updating the D-HOFC by chain products*** — GC derives two incremental HOFCs of length $l$-$c$, say $R_0$ and $R_1$, respectively from two randomly-chosen seeds $r_{0,0}$ and $r_{0,1}$. For any chain $V_0$, we denote its sub-chain confined in time slot interval [$a$, $b$] by $V_0[a, b]$. As illustrated in Figure 4, GC could update sub-chain $V_0[c, l\text{-}1]$ (resp. sub-chain $V_1[c, l\text{-}1]$) by performing a chain product as $V_0'[c, l\text{-}1]=V_0[c, l\text{-}1]*R_0$ (resp. $V_1'[c, l\text{-}1]=V_1[c, l\text{-}1]*R_1$). However, it is not necessary for GC to update whole D-HOFC in such a computationally-intensive way. According to Theorem 1, performing such chain products is equivalent to first computing both the new forward root control key as $v_{c,0}'= v_{c,0}* r_{0,0}$ and the new backward root control key as $v_{l\text{-}1,1}'= v_{l\text{-}1,1}* r_{0,1}$, and then deriving any control key on $V_0'[c, l\text{-}1]$ (resp. $V_1'[c, l\text{-}1]$) from $v_{c,0}'$ (resp. $v_{l\text{-}1,1}'$) when needed; (2) ***Sending rekey messages*** — GC derives $u_i$'s new backward control key $v_{l\text{-}d\text{-}1,1}'$ from $v_{l\text{-}1,1}'$ (note that $u_i$'s new control pair is $\{v_{c,0}',v_{l\text{-}d\text{-}1,1}'\}$) and then sends the following message: $GC\rightarrow u_i$: $c,d,\{v_{c,0}',v_{l\text{-}d\text{-}1,1}'\}$. With this control pair, $u_i$ is able to derive group keys from $K_c'$ to $K_d'$. For the remaining members, GC simply sends the following message by multicast:

$$\text{GC}\Rightarrow U\text{: } c,d,\ \{r_{0,0}, r_{0,1}\}_{v_{l-c-1,1}}\ .$$

Note that every member who is granted access to any group key of the key sequence from $K_c$ to $K_d$ can derive $v_{l\text{-}c\text{-}1,1}$. Therefore, these members can extract $r_{0,0}$ and $r_{0,1}$ from this message, update their control key pair by multiplying them by the corresponding incremental keys derived from either $r_{0,0}$ or $r_{0,1}$ according to Theorem 1, and then compute their new authorized group keys as illustrated in Figure 4.

If the time when another former evictee rejoins after time slot $c$ and the new authorized time slot interval assigned to it, say [$r$, $s$], can be a priori known when $u_i$ rejoins the group, then GC only needs to update sub-chains $V_0[c, \gamma\text{-}1]$ and $V_1[c, \gamma\text{-}1]$ instead of $V_0[c, l\text{-}1]$ and $V_1[c, l\text{-}1]$. Thus, a lot of computational overhead will be saved for both GC and end user.

### 4.1.2 Improved rejoin rekeying

In above rejoin rekeying, the computational overhead of each user is $O(l)$ modular multiplications if the next rejoin event is unpredictable. This is a huge computational overhead for end users when $l$ is big. We provide the following algorithm that achieves efficiency comparable to the original DHC protocol for end users by transferring most amount of computational overhead from end user side to GC side. Suppose $u_i$ rejoins the group and GC assigns it a new authorized time slot interval [$c$, $d$]. Also suppose that when another former evictee rejoins after time slot $c$, the new authorized time slot interval assigned to it is $[r, s]$. We denote an HOWP by $f$. GC first generates a random seed $r_{0,0}$ (resp. $r_{l-c-1}$) for deriving the forward chain (resp. the backward chain in a reverse direction). For each time slot $j$ where $c \le j \le \gamma$, GC first derives the corresponding forward incremental seed $r_{j-c,0}$ from its predecessor $r_{j-c-1,0}$, and the backward incremental seed $r_{l-j-1}$ reversely from its successor $r_{l-j}$ as $r_{l-j-1} = f^{-1}(r_{l-j})$, and then sends the following message by multicast at the beginning of the $j$-th time slot:

$$\mathrm{GC} \Rightarrow U: j, \left\{ r_{j-c,0}, r_{l-j-1,1} \right\}_{v_{l-c-1,1}} .$$

We call it the $j$-th rekey message for short. GC (or multicast sender) may transmit the $j$-th rekey message by piggybacking it on the application data transmitted during the $j$-th time slot. Upon receipt of this message, every legitimate user who is granted access to the $j$-th time slot can extract both incremental seeds from it and compute the corresponding group key $K_j$ as

$$K_j = (v_{j,0} * r_{j-c,0}) \oplus (v_{l-j-1,1} * r_{l-j-1,1}) .$$

Compared to above rejoin rekeying algorithm, for a user with authorized time slot interval $[\alpha, \alpha+m]$ ($c \le \alpha < \alpha+m \le \gamma$), its total overhead for computing authorized group keys is reduced from ($l$-$c$+$m$+1) modular multiplications plus 2 decryptions to $2m$ decryptions plus $4m$ modular multiplications.

Interestingly, the improved rekeying algorithm has a nice property called *self-healing* [34] in the sense that a user can obtain missing group keys on its own without requesting a retransmission from GC. Denote the $i$-th rekey message by $M_i$. Suppose that a user $u$ with authorized time slot interval $[\alpha, \beta]$ ($c \le \alpha < \beta \le \gamma$) got both $M_i$ ($\alpha < i < \beta$) and $M_{j+1}$ ($i < j < \beta$). Then even if all rekey messages within time slot interval $[i+1, j]$ are missing, $u$ is still able to recover all group keys within time slot interval $[i+1, j]$, because $u$ can derive all forward incremental seeds within time slot interval $[i+1, j]$ from $r_{i-c,0}$ extracted from $M_i$, and all backward incremental seeds within time slot interval $[i+1, j]$ from $r_{l-j-2,1}$ extracted from $M_{j+1}$.

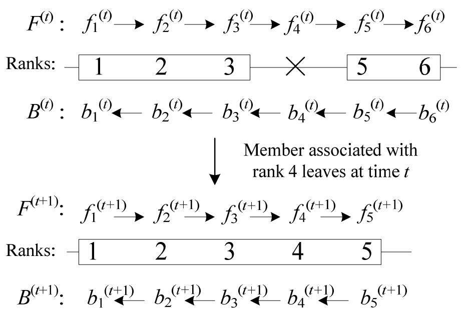
Existing HOWPs (e.g., Rabin functions or RSA functions) are usually trapdoor functions. D-HOFC based on them can be extended in a reverse direction by GC, if it knows the trapdoor information. Thanks to that, for Protocol I, the lifetime of a group can be extended as needed. Of course, GC must protect the private trapdoor information from every group members.

***Remark 1:*** Compared with the DHC protocol, Protocol I is stateful. Every member must remain online in order to ensure receiving each rekey message and updating its control pair accordingly, otherwise it will be unable to compute any future group key.

***Remark 2:*** To save GC from storing and managing a long history revocation list, one solution is to let GC update sub-chains of D-HOFC whenever a member (no matter whether it is a fresh new one or a former evictee) joins the group.

## 4.2 The Stateful User-Based MKD Protocol — Protocol II

In this section, we use D-HOFC to design a stateful user-based MKD protocol. One of the experimental results given in Section 6.4 shows that it has a lower computational overhead for GC than the corresponding DHC-based protocol in 60% time of a MBone audio session. The idea is that when a member joins/leaves the group, GC extends/contracts the D-HOFC accordingly, updates them by chain products, and then broadcasts an encrypted rekey message containing the new group key and two incremental root keys such that all current members except the joining/departing member can extract them and then update their control pairs accordingly using these incremental root keys. In this way, both group forward and backward secrecy are ensured.



$F^{(t)}$: $f_1^{(t)}$ $f_2^{(t)}$ $f_3^{(t)}$ $f_4^{(t)}$ $f_5^{(t)}$ $f_6^{(t)}$
Ranks: 1 2 3 5 6
$B^{(t)}$: $b_1^{(t)}$ $b_2^{(t)}$ $b_3^{(t)}$ $b_4^{(t)}$ $b_5^{(t)}$ $b_6^{(t)}$
Member associated with rank 4 leaves at time $t$
$F^{(t+1)}$: $f_1^{(t+1)}$ $f_2^{(t+1)}$ $f_3^{(t+1)}$ $f_4^{(t+1)}$ $f_5^{(t+1)}$
Ranks: 1 2 3 4 5
$B^{(t+1)}$: $b_1^{(t+1)}$ $b_2^{(t+1)}$ $b_3^{(t+1)}$ $b_4^{(t+1)}$ $b_5^{(t+1)}$

Fig. 5 Leave rekeying based on D-HOFC

Referring to Figure 5, we denote the forward HOFC derived from a root key $f_1^{(t)}$ at time $t$ by $F^{(t)}$ and the backward HOFC derived from a root key $b_n^{(t)}$ at time $t$ by $B^{(t)}$. The control pair associated with rank $k$ at time $t$ are denoted by $\{f_k^{(t)}, b_k^{(t)}\}$. Given an arbitrary D-HOFC, its ranks can be deleted, added, and reassigned unlike a DHC. Suppose that the current length of a D-HOFC defined over a HOWP $g$ is $n$. When a member $u_i$ with rank $k$ ($1 \le k \le n$) leaves the group at time $t$, GC performs the following steps to complete leave rekeying: (1) ***Contracting and updating the D-HOFC*** — GC generates two random incremental root keys $r_{1,0}^{(t)}$ and $r_{n-1,1}^{(t)}$, and computes a new forward root control key (resp. a new backward root control key) as $f_1^{(t+1)} = g(f_1^{(t)}) * r_{1,0}^{(t)}$ (resp. $b_{n-1}^{(t+1)} = g(b_n^{(t)}) * r_{n-1,1}^{(t)}$). GC is now able to derive a new ($n$-1)-length D-HOFC consisting of a new forward chain $F^{(t+1)}$ and a new backward chain $B^{(t+1)}$. The former is derived from $f_1^{(t+1)}$ and the latter from $b_{n-1}^{(t+1)}$. Referring to Figure 5, the effect of such operations is equivalent to contracting the original D-HOFC by one rank through chain products; (2) ***Multicasting a rekey message*** — GC derives $f_{k-1}^{(t)}$ and $b_{k+1}^{(t)}$ respectively from $f_1^{(t)}$ and $b_n^{(t)}$ and then generates a new group key $GK^{(t+1)}$. At last, GC sends the following rekey message by multicast:

$$GC \Rightarrow S^{(t+1)}: \quad k, \quad \left\{ GK^{(t+1)}, r_{1,0}^{(t)}, r_{n-1,1}^{(t)} \right\}_{f_{k-1}^{(t)}} \ (if \ k < n) \ , \left\{ GK^{(t+1)}, r_{1,0}^{(t)}, r_{n-1,1}^{(t)} \right\}_{b_{k+1}^{(t)}} \ (if \ k > 1) \qquad (1)$$

where $S^{(t+1)} = S^{(t)} - u_i$.

Every member except $u_i$ can extract $GK^{(t+1)}$, $r_{1,0}^{(t)}$ and $r_{n-}$

$_{1,1}{}^{(t)}$ from either the first part or the second part of this message, and then run Algorithm I to update their control pairs. Any former evictee cannot decrypt the above message, because its control pair had been invalidated by a chain update operation when it left the group and thus it can derive neither of the two control keys used to encrypt the rekey message.

```
ALGORITHM I — UPDATING CONTROL PAIR
For an arbitrary member $u_j$ ($j \neq i$), suppose its associated rank is $a$ ($a \neq k$),
if $a<k$, then
  $u_j$ computes its new control pair as
    $f_a^{(t+1)} = g(f_a^{(t+1)})*r_{a,0}^{(t)}$ and
    $b_a^{(t+1)} = b_a^{(t)}*r_{a,1}^{(t)}$;
else (i.e., $a>k$),
  $u_j$ first reduces its rank by one and then computes its new control pair as
    $f_{a\text{-}1}^{(t+1)} = f_a^{(t)}*r_{a\text{-}1,0}^{(t)}$ and
    $b_{a\text{-}1}^{(t+1)} = g(b_a^{(t)})*r_{a\text{-}1,1}^{(t)}$.
```

For leave rekeying, the computational cost of GC includes 6 encryptions, $n$-1 evaluations of an HOWP for computing encryption keys for rekey message, and 2 evaluations of an HOWP plus 2 modular multiplications for updating both root control keys. The computational cost of each group member includes $n$-1 evaluations of an HOWP for computing decryption keys for rekey message in worst case, 3 decryptions, and $n$-1 evaluations of an HOWP plus 2 modular multiplications for updating its control keys. The multicast size includes 6 encrypted keys. Note that we do not count the bandwidth cost of user *id* or rank *id* in a rekey message.

Suppose the current length of D-HOFC is $n$. When a new member $u_i$ joins the group at time $t$, GC performs the following steps to complete join rekeying: (1) ***Extending the D-HOFC*** — GC adds a new rank $n$+1 at the tail of the D-HOFC, and computes its associated control pair as $b_{n+1}{}^{(t)}=g^{-1}(b_n{}^{(t)})$ and $f_{n+1}{}^{(t)}=g(f_n{}^{(t)})$. The effect of such an operation is equivalent to extending the original D-HOFC at its tail by one rank; (2) ***Updating the extended D-HOFC by chain products*** — GC generates two random incremental root keys $r_{1,0}{}^{(t)}$ as well as $r_{n+1,1}{}^{(t)}$, and compute a new forward root control key as $f_1{}^{(t+1)}=f_1{}^{(t)}*r_{1,0}{}^{(t)}$ as well as a new backward root control key as $b_{n+1}{}^{(t+1)}=g^{-1}(b_n{}^{(t)})*r_{n+1,1}{}^{(t)}$. With these new root control keys, GC can derive a new ($n$+1)-length D-HOFC; (3) ***Sending rekey messages*** — GC generates a new group key $GK^{(t+1)}$ and sends the following message: $GC \rightarrow u_i$: $n$+1, $GK^{(t+1)}$, $\{f_{n+1}{}^{(t+1)}, b_{n+1}{}^{(t+1)}\}$. For members except $u_i$, GC simply sends the following rekey message by multicast:

$$GC \Rightarrow S^{(t+1)}:\ n+1, \left\{GK^{(t+1)}, r_{1,0}{}^{(t)}, r_{n+1,1}{}^{(t)}\right\}_{GK^{(t)}}.$$

Every members except $u_i$ can decrypt this message to obtain the new group key $GK^{(t+1)}$, $r_{1,0}{}^{(t)}$ and $r_{n+1,1}{}^{(t)}$, and update their control keys by multiplying them by the corresponding incremental keys deriving from either $r_{1,0}{}^{(t)}$ or $r_{n+1,1}{}^{(t)}$.

For join rekeying, the computational cost of GC includes 3 encryptions, 1 calculation of modular square root, 1 modular multiplication for updating the forward root control key, and 2 modular multiplications plus $n$ evaluations of an HOWP for computing the control key pair for the joining member. The computational cost of each group member includes 3 decryptions, and $n$ evaluations of an HOWP plus 2 modular multiplications for updating its control pair. The multicast size includes 3 encrypted keys and unicast size includes 3 keys.

Protocol II is vulnerable to the same member collusion attack as the LORE protocol. But for non-member collusion attack, a coalition between a former evictee and a later-joining member is useless. Suppose that a user $u_i$ with rank $k$ left the group at time $t$, and a user $u_j$ joined at time $t$+1 and was assigned with rank $n>k$. The rekey message $M^{(t)}$ transmitted by GC when $u_i$ left at time $t$ is as (1). The first part (resp. the second part) of $M^{(t)}$ was encrypted by a forward control key $f_{k\text{-}1}{}^{(t)}$ (resp. $b_{k+1}{}^{(t)}$). When $u_j$ joined the group at time $t$+1, $u_j$ was provide by GC with control key $\{f_n{}^{(t+2)}, b_n{}^{(t+2)}\}$. According to Protocol II, the forward control keys $f_{k\text{-}1}{}^{(t)}$ and $f_n{}^{(t+2)}$ (resp. backward control keys $b_{k+1}{}^{(t)}$ and $b_n{}^{(t+2)}$) belong to different forward chains $F^{(t)}$ and $F^{(t+2)}$ (backward chains $B^{(t)}$ and $B^{(t+2)}$), respectively. Since neither of $f_{k\text{-}1}{}^{(t)}$ and $b_{k+1}{}^{(t)}$ can be derived from the collective control key pairs of $u_i$ and $u_j$, i.e., $\{f_k{}^{(t)}, b_k{}^{(t)}, f_n{}^{(t+2)}, b_n{}^{(t+2)}\}$, both $M^{(t)}$ (containing $GK^{(t+1)}$) and $M^{(t+1)}$ (encrypted by $GK^{(t+1)}$) are indecipherable to $u_i$ and $u_j$.

## 4.3 The EKT+ Protocol

In this section, we first prove the existence of a tree blinding operation for a particular type of TD-HOFT called *Exclusive Key Tree* (EKT). Then we introduce group rekeying algorithms based on tree blinding and tree product.

### *4.3.1 Existence of a tree blinding operation for EKTs*

We first introduce HOWPs used by the EKT protocol, and then prove the existence of a tree blinding operation for EKTs. Given two Blum integers with the same bit lengths, say $m_L$ and $m_R$, we use $m_{LR}$ to denote $m_{LR} = m_L*m_R$. The two HOWPs $f_L(x)$ and $f_R(x)$ used by the EKT protocol are defined as $f_L(x) = (x^2 \bmod m_{LR}) \bmod m_L$, and $f_R(x) = (x^2 \bmod m_{LR}) \bmod m_R$, respectively. We denote the key associate with a node $n_i$ by $EK_i$ in below. GC chooses a random root seed $EK_1 \in Z^*_{m_{LR}}$ and derives the whole key tree using $f_L(x)$ and $f_R(x)$ in a top-down manner as discussed in Section 2.2. Recall that each leaf key $EK_i$ is called an exclusive key for the user associated with $n_i$. In this sense, we call a TD-HOFT derived in the above manner an *exclusive key tree* (EKT). It is readily seen that given an arbitrary EKT, its leaf keys satisfy the two sufficient conditions for a TD-HOFT to be used as an access control structure: *collision-freeness* and *independence*.

***Theorem 3:*** *For an arbitrary EKT X, a tree product of X and itself (i.e., self tree product) is a tree blinding operation.*

Proof: For an arbitrary EKT $X$, let $Y = X*X$. According to Theorem 2, $Y$ is also an EKT. For an arbitrary node $x_i$ (resp. $y_i$) in an EKT $X$ (resp. $Y$), we denote its left child and right child by $x_{2i}$ (resp. $y_{2i}$) and $x_{2i+1}$ (resp. $y_{2i+1}$), respectively. According to Definition 5, $y_1$ is computed as $y_1 = x_1{}^2 \bmod m_{LR}$. Given $i$>1, if $i$ is even, $y_i$ is computed as $y_i = x_i{}^2 \bmod m_L$, otherwise $y_i = x_i{}^2 \bmod m_R$. Because all three modular square functions are one-way, it is computationally infeasible to compute any node of $X$ from all nodes of $Y$. Therefore, according to Definition 6, a self-tree product is a tree blinding operation. □

### 4.3.2 Group rekeying algorithms

We improve the EKT protocol from the following two aspects: (1) introducing simple operations for adding/deleting nodes to/from an EKT; (2) using a tree blinding operation to design a join rekeying algorithm that improves the computational performance of the EKT protocol. We call the improved protocol EKT+.

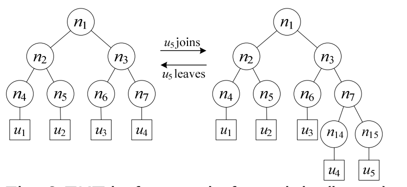


Fig. 6 EKT before and after a join (leave)

First recall that the personal key of each user, say $u_i$, includes those keys associated with siblings of those nodes in its path to the root. For convenience, we call these keys *sibling keys* of $u_i$. Denote by $EK_i^{(t)}$ the key associated with node $n_i$ at time $t$. For example, referring to Figure 6, $u_5$ got three sibling keys $EK_2^{(t)}$, $EK_6^{(t)}$ and $EK_{14}^{(t)}$ when it joined the group at time $t$.

When a new member $u_i$ joins the group at time $t$, GC performs the following steps to complete join rekeying: (1) ***Performing a tree-growth operation (as illustrated in Figure 6)*** — GC first finds a shallowest leaf node, say $n_k$ (suppose it is currently associated with user $u_j$), then adds a left child node $n_{2k}$ and a right child node $n_{2k+1}$ to $n_k$, and derives their corresponding exclusive keys $EK_{2k}^{(t)}$ and $EK_{2k+1}^{(t)}$ as $EK_{2k}^{(t)}=f_L(EK_k^{(t)})$ and $EK_{2k+1}^{(t)}=f_R(EK_k^{(t)})$, respectively. After those operations, GC associates users $u_j$ and $u_i$ with $n_{2k}$ and $n_{2k+1}$, respectively; (2) ***Performing a tree blinding operation*** — To ensure group backward secrecy, GC needs to perform a tree blinding operation on the expanded key tree. For an arbitrary exclusive key $EK_i^{(t)}$, we denote its blinded version by $EK_i^{(t+1)}$. In fact, it is not necessary for GC to actually perform a tree blinding operation on the whole key tree. According to Theorem 3, GC only needs to compute its blinded root exclusive key $EK_1^{(t+1)}$ as $EK_1^{(t+1)}= (EK_1^{(t)})^2 \bmod m_{LR}$, and then derive those sibling keys for the joining member $u_i$ from $EK_1^{(t+1)}$; (3) ***Sending rekey messages*** — GC derives the new group key $GK^{(t+1)}$ from current group key $GK^{(t)}$ as $GK^{(t+1)}=h(GK^{(t)})$ where $h$ is a public hash function, and sends the following messages:

GC→ $u_i$: $n_{2k+1}$,$GK^{(t+1)}$, *sibling keys for* $u_i$,

GC→ $u_j$: $n_{2k+1}$,$EK_{2k+1}^{(t+1)}$.

For remaining members, GC simply sends the following message by multicast:

GC⇒$S^{(t+1)}$: a *rekey notification*.

After receiving this notification, all members except $u_i$ can update the group key as GC does, and update their own sibling keys by modular square operations as described in the proof of Theorem 3.

For join rekeying, the computational cost of GC includes 1 evaluation of a hash function for computing the updated group key, 1 evaluations of an HOWP for updating the root exclusive key, and $2\log_2 n$ evaluations of an HOWP for computing all blinded sibling keys for the joining member and a new blinded sibling key for the sibling of the joining member. The computational cost of each group member includes 1 evaluation of a hash function for computing the updated group key and $\log_2 n$ evaluations of an HOWP for updating its sibling key. The multicast size includes one notification message and unicast size includes $\log_2 n+2$ keys.

When a member $u_i$ (suppose that it is associate with node $n_{2k+1}$) leaves the group at time $t$, GC performs the following steps to complete leave rekeying: (1) ***Performing a tree-contraction operation (as illustrated in Figure 6)*** — GC deletes both $n_{2k+1}$ and its sibling $n_{2k}$, and then associates the sibling user of $u_i$ (who was associated with node $n_{2k}$ ) with $n_k$; (2) ***Updating the shrunk key tree by a tree product*** — GC generates a random incremental root seed $r_1^{(t)}$. From $r_1^{(t)}$, it could derive the whole incremental key tree $T$ in a top-down manner. Now, to ensure group forward secrecy, GC may update the shrunk key tree by performing a tree product of it and $T$. However, it is not necessary for GC to update the whole shrunk key tree in such way. According to Theorem 2, GC only needs to compute the updated root exclusive key $EK_1^{(t+1)}$ as $EK_1^{(t+1)}= EK_1^{(t)}* r_1^{(t)}$, and then derive the exclusive key $EK_{2k+1}^{(t)}$ for the departing member $u_i$ from $EK_1^{(t)}$; (3) ***Multicasting a rekey message*** — GC generates a random new group key $GK^{(t+1)}$ and sends the following rekey message by multicast:

GC⇒$S^{(t+1)}$: $n_{2k+1}$, $\left\{GK^{(t+1)}, r_1^{(t)}\right\}_{EK_{2k+1}^{(t)}}$ where $S^{(t+1)}=S^{(t)}-u_i$.

Every member except $u_i$ can extract $GK^{(t+1)}$ and $r_1^{(t)}$ from this message, and then update its own sibling keys by multiplying them by their corresponding incremental key derived from $r_1^{(t)}$. For example, if $i$ is even, $EK_i^{(t+1)}$ is computed as $EK_i^{(t+1)}= EK_i^{(t)}*r_i^{(t)} \bmod m_L$.

For leave rekeying, the computational cost of GC includes 2 encryptions, 1 modular multiplication for computing updated root exclusive key, and $\log_2 n$ evaluations of an HOWP for computing the encryption key for rekey message. The computational cost of each group member includes $\log_2 n$ evaluations of an HOWP for computing decryption key for rekey message in worst case, 2 decryptions, $2\log_2 n$ evaluations of an HOWP for computing the incremental keys corresponding to its sibling keys, and $\log_2 n$ modular multiplications for updating its sibling keys. The multicast size includes 2 encrypted keys.

Both the EKT and EKT+ protocols are vulnerable to the same member collusion attack as the LW protocol. But for non-member collusion attack, using a similar argument as given in the end of Section 4.2, it is readily seen that a coalition between a former evictee and a later-joining member is useless for both the EKT and EKT+ protocols. Both the EKT and EKT+ protocols can be easily extended to support batch group rekeying by using one of the subset-cover techniques [13] as Liu and Wang dealt with the LW protocol [23].

## 4.4 From a 1-Resilient MKD Protocol to a Hybrid MKD Protocol with Collusion-Bandwidth Tradeoffs

According to a recent research result by Liu and Wang

[23], all personal key assignment algorithms based on either D-OFC (including DHC and D-HOFC) or TD-OFT (including BHT and TD-HOFT) inherently suffer from member collusion attack because all of them are based on a concept called *exclusive key*. Therefore, stateful user-based MKD protocols based on these personal key assignment algorithms are bound to be 1-resilient. However recent research results [23], [28] clarify the significance of research on 1-resilient MKD protocols by showing that any 1-resilient MKD protocol with constant communication overhead can be used in conjunction with a collusion-resistant tree-based protocol (e.g., the LKH protocol [7]) to construct a hybrid protocol with tunable collusion-bandwidth tradeoffs. Since the resulting hybrid protocol has a lower communication overhead than the corresponding collusion-resistant protocol, it is more suitable for cost-sensitive applications or resource-constrained environments.

The idea of constructing a hybrid stateful protocol is simply as follows. A group of receivers are divided into divisions. A key called *division key* is shared by all receivers in a division. GC simply employs an independent instance of a 1-resilient stateful MKD protocol for each division to rekey its division key. While each division is regarded as a leaf node in an LKH key tree, GC uses the LKH protocol to rekey group keys across all divisions. It is readily seen that a group of *k* colluding receivers can succeed if and only if at least two of them are allocated to the same division since combining control keys from distinct divisions does not help the colluders. To reduce vulnerability to collusion attack, we must increase the number of divisions, say *d*. But a greater *d* means a higher communication overhead. Therefore, there exists a tradeoff between collusion resistance and bandwidth. To achieve the highest level of collusion protection, GC has to allocate a receiver to a randomly-chosen division when it joins the group. However, if we choose to use a 1-resilient stateful MKD protocol based on static group access structures (DHC or BHT) for constructing a hybrid protocol, the following problem is likely to arise. Because a static group access structure has a pre-specified limit on its size, and cannot be extended or expanded after it is put into use. When a joining member is randomly allocated to a certain division by GC, all free ranks (for DHC) or free leaf key nodes (for BHT) with respect to this division have been used up, and GC has to reallocate this receiver to another division which has free ranks or leaf key nodes. Thus, the randomness of allocating receivers to divisions is compromised, and so is the security of the resulting hybrid protocol. But 1-resilient stateful MKD protocols based on dynamic group access structures (D-HOFC or TD-HOFT) don't have this problem because dynamic group access structures can be extended (or expanded) as needed. Therefore, Protocol II and the EKT+ protocol are more suitable for constructing a hybrid stateful protocol with tunable collusion-bandwidth tradeoffs than their counterparts — the LORE protocol and the LW protocol.

## 5 Proofs of Security (1-Resilience)

Panjwani [35] developed a symbolic security model for analysing generic user-based MKD protocols. In this model, all keys and messages generated by a user-based MKD protocol are treated as abstract data types and cryptographic primitives as abstract functions over such data types. Security can be specified by *recoverability* in the sense that some group key is safe if it cannot be recovered by adversaries from their aggregated personal keys and all rekey messages. Panjwani proves security of two variants of the LKH protocol [8],[7],[9], the complete sub-tree protocol [13] and the subset difference protocol [13] using a straightforward inductive argument in this model. Below, we prove security of our user-based MKD protocols (Protocol II and the EKT+ protocol) under this model. We also extend Panjwani's model to support time-based MKD protocols and prove security of our time-based MKD protocol (Protocol I) under the extended model.

Consider a multicast group with a lifetime of *t* time slots, labelled by 0, 1,⋯, *t*-1. For a *t*-time-slot MKD protocol Π, we introduce the following notations. The group key used to encrypt all data transmitted during the *i*-th time slot is denoted by $K^{(i)}$. A rekey message generated by protocol Π at the beginning of the *i*-th time slot is denoted by $M_i^{\Pi}$. Let $M_{\vec{t}}^{\Pi}$ denote the set of all the rekey messages generated by protocol Π up to the *t*-th time slot.

Consider a multicast group of *n* users, labelled by 1, 2,⋯, *n*. For an *n-user MKD protocol* Π, we introduce the following notations given by [35]. At any time *t*, the privileged set of users who are authorized to receive data sent over a multicast channel is denoted by $S^{(t)} \subseteq \{1,2,\ldots, n\}$. The rekey message generated by protocol Π for $S^{(t)}$ is denoted by $M_{S^{(t)}}^{\Pi}$. The group key used to encrypt all the data sent to $S^{(t)}$ is denoted by $K^{(t)}$. Let $[n]$ denote the set $\{1,\cdots,n\}$ and $2^{[n]}$ denote the power set of $[n]$. An arbitrary group dynamics up to time *t* can be uniquely represented by a contiguous sequence of privileged user sets $\overrightarrow{S^{(t)}} = (S^{(0)}, S^{(1)}, \cdots, S^{(t)}) \in (2^{[n]})^t$. A sequence $\overrightarrow{S^{(t)}} \in (2^{[n]})^t$ is called *simple*, if for all $t \geq 1$, $S^{(t-1)}$ changes into $S^{(t)}$ through a single change in group membership. Let $M_{\overrightarrow{S^{(t)}}}^{\Pi}$ denote the set of all the rekey messages generated by protocol Π up to time *t*, i.e., $M_{\overrightarrow{S^{(t)}}}^{\Pi} = \bigcup_{1 \leq t' \leq t} M_{S(t')}^{\Pi}$.

For both time-based MKD protocols and user-based MKD protocols, we use the following notations. Each user *i* obtains a personal key set $PKS_i$ from GC when it joins the group. For any message-set *M*, we use *Rec*(*M*) to denote the set of all messages that are recoverable from it by using all sorts of cryptographic transformations employed by a MKD protocol (irrespective of the number of steps required to do so).

***Definition 7:*** An *l*-time-slot MKD protocol Π is called *secure against single-user attacks* (i.e., *1-resilient*), if for any user *i* with authorized time slot interval [*α*, *β*] where *β*≤*l*, $\forall t \notin [\alpha, \beta], K^{(t)} \notin \mathrm{Re}c(PKS_i \cup M_{\vec{l}}^{\Pi})$.

***Definition 8:*** An $n$-user immediate rekeying MKD protocol $\Pi$ is called *1-resilient*, if for all $t\geq 0$, and all simple sequence $\overrightarrow{S^{(t)}} \in (2^{[n]})^t$, $\forall i \notin S^{(t)}$, $K^{(t)} \notin \mathrm{Re}c(PKS_i \cup M^{\Pi}_{\overrightarrow{S^{(t)}}})$.

It is easy to derive that 1-resilience implies both group forward secrecy against single-user attacks and group backward secrecy against single-user attacks.

***Definition 9:*** An $l$-time-slot MKD protocol $\Pi$ is called *correct*, if for any user $i$ with authorized time slot interval $[\alpha, \beta]$ where $\beta\leq l$, $\forall t \in [\alpha, \beta]$, $K^{(t)} \in \mathrm{Re}c(PKS_i \cup M^{\Pi}_{\overrightarrow{l}})$.

***Definition 10:*** An $n$-user EKT-based MKD protocol $\Pi$ is called *correct*, if for all $t\geq 0$, and all simple sequence $\overrightarrow{S^{(t)}} \in (2^{[n]})^t$, $\forall i\in S^{(t)}$, $i$ always knows $K^{(t)}$ and its sibling keys in $Tr^{(t)}$ where $Tr^{(t)}$ denotes the key tree corresponding to $S^{(t)}$, and no other keys in $Tr^{(t)}$.

The correctness of Protocol I is obvious. Below, we only prove security of the rejoin rekeying algorithm of Protocol I. Security of the improved version can be proved using a similar argument.

***Theorem 4:*** *Protocol I (rejoin rekeying) is correct and 1-resilient.*

Proof: Without loss of generality, we only need to consider two types of users: (1) an arbitrary user who joined the group only once; (2) an arbitrary user who joined the group twice.

*Case (1):* Consider an arbitrary user $i$ who joined the group only once and whose authorized time slot interval was $[k, h]$. Referring to Figure 4, according to Protocol I, $PKS_i=\{v_{k,0}, v_{l-h-1,1}\}$. For any $t\leq h$, consider an arbitrary rekey message $M_t^l$ that is supposed to be used to update a chain of group keys corresponding to time slot interval $[t, s]$. Without loss of generality, also suppose that there is no other rejoin event that happens during time slot interval $[t, h]$. According to Protocol I, $M_t^l$ is encrypted under the intermediate seed $v_{l-t-1,1}$. User $i$ can derive $v_{l-t-1,1}$ from $v_{l-h-1,1}$, and then use it to decrypt $M_t^l$ to obtain those incremental seeds. However, with these seeds and $PKS_i$, $i$ is only able to update those group keys belonging to time slot interval $[t, s]\cap[k, h]$. For any $t>h$, consider an arbitrary rekey message $M_t^l$. According to Protocol I, $M_t^l$ is encrypted under $v_{l-t-1,1}$. User $i$ cannot derive $v_{l-t-1,1}$ from $v_{l-h-1,1}$ in a reverse direction (referring to Figure 4). Therefore, $M_t^l$ is useless to $i$. To sum up, we have $\forall t \notin [k,h]$, $K^{(t)} \notin \mathrm{Re}c(PKS_i \cup M^{\mathrm{I}}_{\overrightarrow{l}})$.

*Case (2):* Consider an arbitrary user $i$ whose first authorized time slot interval is $[\alpha, \beta]$ and second authorized time slot interval is $[\gamma, \delta]$. According to Protocol I, the aggregated personal key set is $PKS_i =\{v_{\alpha,0}, v_{l-\beta,1}, v_{\gamma,0}', v_{l-\delta,1}'\}$. Note that control pairs $\{v_{\alpha,0}, v_{l-\beta,1}\}$ and $\{v_{\gamma,0}', v_{l-\delta,1}'\}$ belong to two distinct D-HOFC, respectively. Applying the same argument as Case (1) to time slot interval $[0, \gamma-1]$ with respect to the first D-HOFC, we have $\forall t \in [0,\alpha-1]\cup[\beta+1,\gamma-1]$, $K^{(t)} \notin \mathrm{Re}c(PKS_i \cup M^{\mathrm{I}}_{\overrightarrow{\gamma-1}})$. Applying the same argument as Case (1) to time slot interval $[r, l-1]$ with respect to the second D-HOFC, we have $\forall t \in [\delta+1, l-1]$, $K^{(t)} \notin \mathrm{Re}c(PKS_i \cup M^{\mathrm{I}}_{\overrightarrow{l}})$. In all, we have $\forall t \notin [\alpha,\beta]\cup[\gamma,\delta]$, $K^{(t)} \notin \mathrm{Re}c(PKS_i \cup M^{\mathrm{I}}_{\overrightarrow{l}})$. □

***Theorem 5:*** *The EKT+ protocol is correct and 1-resilient.*

Proof: In fact, we can prove an even stronger claim that for all $t\geq 0$, and all simple sequence $\overrightarrow{S^{(t)}} \in (2^{[n]})^t$, $\forall i\notin S^{(t)}$, $K^{(t)} \cup Tr^{(t)} \notin \mathrm{Re}c(PKS_i \cup M^{EKT+}_{\overrightarrow{S^{(t)}}})$. We prove it using induction over $t$. For $t$=0, since $S^{(0)}=\varnothing$, the claim is trivially true. Now we argue that if the claim is true for some $t\geq 1$, then it is true for $t$+1 as well. For any simple sequence $\overrightarrow{S^{(t+1)}} = (S^{(0)}, S^{(1)}, \cdots, S^{(t)}, S^{(t+1)})$, we only need to consider the following cases:

Case 1 ($i\in S^{(t)} \wedge i\in S^{(t+1)}$, and $S^{(t)}$ changes into $S^{(t+1)}$ due to another member's departure at time $t$): According to the leave rekeying algorithm of the EKT+ protocol, $i$ can recover the root incremental seed $r_1^{(t)}$ and group key $K^{(t+1)}$ from rekey message $M^{EKT+}_{S^{(t+1)}}$. From inductive hypothesis, $i$ only holds $K^{(t)}$ and those sibling keys in $Tr^{(t)}$ as required by Definition 10. From all incremental seeds (derived from $r_1^{(t)}$) and those sibling keys in $Tr^{(t)}$, it can compute and only compute those sibling keys in $Tr^{(t+1)}$ by multiplying an sibling key $EK_i^{(t)}$ in $Tr^{(t)}$ by its corresponding incremental seed $r_i^{(t)}$.

Case 2 ($i\in S^{(t)} \wedge i\in S^{(t+1)}$, and $S^{(t)}$ changes into $S^{(t+1)}$ due to another member's join at time $t$): According to the join rekeying algorithm of the EKT+ protocol, $i$ can recover nothing from a rekey notification message. From inductive hypothesis, $i$ only holds $K^{(t)}$ and those sibling keys in $Tr^{(t)}$ as required by Definition 10. It can derive $K^{(t+1)}$ as $K^{(t+1)}=h(K^{(t)})$, and those sibling keys as $EK_i^{(t+1)}= (EK_i^{(t)})^2 \bmod m_L$ (if $i$ is even) or $EK_i^{(t+1)}= (EK_i^{(t)})^2 \bmod m_R$ (if $i$ is odd).

Case 3 ($i\notin S^{(t)} \wedge i\in S^{(t+1)}$): That is to say, $i$ joins the group at time $t$. According to the join rekeying algorithm of the EKT+ protocol, every newly joining member $i$ can recover just the same key materials as required by Definition 10 from the rekey message $M^{EKT+}_{S^{(t+1)}}$ (note that $M^{EKT+}_{S^{(t+1)}}$ includes both a unicast message and a multicast message sent by GC).

Case 4 ($i\in S^{(t)} \wedge i\notin S^{(t+1)}$): That is to say, $i$ is evicted at time $t$. From the inductive hypothesis, all secrets that $i$ knows are $K^{(t)}$ and those sibling keys in $Tr^{(t)}$. According to the leave rekeying algorithm of the EKT+ protocol, $i$ can recover neither $K^{(t+1)}$ nor the root incremental seed $r_1^{(t)}$ from $M^{EKT+}_{S^{(t+1)}}$. Without $r_1^{(t)}$, $i$ can never compute any sibling key $EK_i^{(t+1)}$ in $Tr^{(t+1)}$.

Case 5 ($i\notin S^{(t)} \wedge i\notin S^{(t+1)}$): That is to say, $i$ is evicted before time $t$. From the inductive hypothesis, $i$ can never recover (or compute) $K^{(t)}$ and any key $EK_i^{(t)}$ in $Tr^{(t)}$. Since $M^{EKT+}_{S^{(t+1)}}$ is encrypted by some exclusive key $EK_i^{(t)}$ in $Tr^{(t)}$, $i$ can recover neither $K^{(t+1)}$ nor the root incremental seed $r_1^{(t)}$ from $M^{EKT+}_{S^{(t+1)}}$. Thus $i$ can compute no key in $Tr^{(t+1)}$. □

***Theorem 6:*** *Protocol II is correct and 1-resilient.*

TABLE 1
COMPARISON OF TIME-BASED MKD PROTOCOLS

| | *Group access structure* | *GC stora.* | *Mem. stora.* | *Total GC comp.* | *Total mem. comp.* | *Unicast size* | *Total multicast size* | *Rejoining mem. attack* | *Self-Healing* | *Extensible group lifetime* |
|---|---|---|---|---|---|---|---|---|---|---|
| **BHC (Auto. rekeying)** | BHC (static) | 2 keys | 2 keys | $2l*C_h$ | $2m*C_h$ | — | — | Yes | — | No |
| **Protocol I (Rejoin rekeying)** | BD-HOFCs (dynamic) | 2 keys | 2 keys | $2C_E + (l-d+1)*C_M$ | $2C_D + (l-c+m+1)*C_M$ | 2 keys | 2 keys | No | No | Yes |
| **Protocol I (improved version)** | | | | $2(l-c-1)*C_E + (l-c-1)*C_M + (l-c-1)*C_{sqr}$ | $2m*C_D + 4m*C_M$ | | $2(l-c)$ keys | | Yes | |

*For all algorithms, we assume that the group's lifetime is l time slots. For Protocol I, we consider its performance in the case that a former evictee rejoins and is assigned by GC with a new authorized time slot interval [c, d].* $C_E$, $C_D$, $C_h$, $C_M$, $C_{Sqr}$ *denote the computation cost of 1 block encryption, 1 block decryption, 1 hash, 1 modular multiplication and 1 modular square root, respectively. Since a Blum-Williams function is a modular square function, its computational cost is almost equivalent to that of a modular multiplication and thus we use* $C_M$ *to denote its computational cost too. To count the computational cost of a member (except the joining one), we assume that it still has m authorized time slots since the c-th time slot.*

TABLE 2
COMPARISON OF USER-BASED MKD PROTOCOLS

| Measures \ Protocols | | LORE | Protocol II | LW | | EKT | EKT+ |
|---|---|---|---|---|---|---|---|
| *Group access structure* | | BHC (static) | BD-HOFCs (dynamic) | BHT (static) | | TD-HOFT (dynamic) | TD-HOFT (dynamic) |
| *GC stora.* | | 2 keys | 2 keys | 1 key | | 1 key | 1 key |
| *Member stora.* | | 2 keys | 2 keys | $log_2N$ keys | | $log_2n$ keys | $log_2n$ keys |
| *Join Rekeying* | *GC comp.* | $1C_E+N*C_h$ | $3C_E + (n+4)*C_M$ | $2log_2N*C_h$ | | $2C_E + (2log_2n+1)*C_M$ | $1C_h+(2log_2n+1)*C_M$ |
| | *Member comp.* | $1C_D$ | $3C_D + (n+2)*C_M$ | $1C_h$ | | $2C_D + (3log_2n-1)*C_M$ | $1C_h+log_2n*C_M$ |
| | *Unicast size* | 3 keys | 3 keys | $log_2N+1$ keys | | $log_2n+1$ keys | $log_2n+2$ keys |
| | *Multicast size* | 1 key | 3 keys | 1 notification | | 2 keys | 1 notification |
| | | | | Stateful | Stateless | | |
| *Leave Rekeying* | *Wst. GC comp.* | $4C_E + (N-1)*C_h$ | $6C_E + (n+3)*C_M$ | $1C_E+log_2N*C_h$ | $1C_E+[2r-2+r*log_2(N/r)]*C_h$ | $2C_E + (log_2n+1)*C_M$ | $2C_E + (log_2n+1)*C_M$ |
| | *Wst. member comp.* | $2C_D+N*C_h$ | $3C_D + 2n*C_M$ | $1C_D+log_2N*C_h$ | $1C_D+[2r-3+r*log_2(N/r)-2log_2r]*C_h$ | $2C_D + 4log_2n*C_M$ | $2C_D + 4log_2n*C_M$ |
| | *Multicast size* | 2 keys | 6 keys | 1 key | | 2 keys | 2 keys |
| *Suitable for constructing a hybird protocol* | | No | Yes | No | | Yes | Yes |
| *Collusion between any two non-members* | | Yes | No | Yes | No | No | |

*Denotations of* $C_E$, $C_D$, $C_h$, $C_M$ *are same as Table 1. In addition, for protocols based on static structures, we use N to denote the static group size which is a pre-specified number of all registered and prospect users, and r to denote the number of all non-members at the time of rekeying. For protocols based on dynamic structures, we use n to denote the dynamic group size which is the number of current group members.*

Theorem 6 can be proved using a similar argument as above.

## 6 PERFORMANCE COMPARISON AND EVALUATION

In this section, we first compare the storage, communication and computation complexity of our proposed MKD protocols with related MKD protocols in terms of the number of keys needed to be stored or transmitted, and the number of times that each cryptographic primitive is called. To evaluate the computational cost of modular multiplication operation of which our protocols makes intensive usage, we make a speed benchmark test with related cryptographic primitives. To further investigate how performance changes with time in practice for every related user-based MKD protocol, we also perform an experiment using MBone user activity data. All those experimental results are presented in Section 6.3 and Section 6.4, respectively.

### 6.1 Comparison of Time-Based MKD Protocols

We summarize those performance-related discussions dispersed in Section 4.1, and provide in Table 1 a comparison between the DHC protocol and Protocol I which covers the following quantitative measures: GC storage requirement, member storage requirement, GC's total computational overhead, total computational overhead of a member, the size of the unicast rekey message, the total size of all multicast rekey messages. Some qualitative measures are also covered in Table 1. According to Table 1, rejoin rekeying algorithm of Protocol I counters the rejoining member attack on the DHC protocol at the cost of a computational complexity of $O(l)$ for end user. The improved rejoin rekeying algorithm greatly reduces this complexity from $O(l)$ to $O(m)$ at the cost of a greater total multicast size and a higher computational overhead for GC.

## 6.2 Comparison of User-Based MKD Protocols

We summarize performance-related discussions dispersed in Sections 4.2 and 4.3, and provide in Table 2 a comprehensive comparison among five stateful user-based MKD protocols including the LORE protocol, Protocol II, the LW protocol, the EKT protocol, and the EKT+ protocol. This comparison covers two qualitative measures (suitability for constructing a hybrid protocol and collusion resistance) and six quantitative measures (GC storage requirement, member storage requirement, GC computational overhead, member computational overhead, unicast size, and multicast size).

According to Table 2, compared with protocols based on chain structures (DHC and D-HOFC), those based on tree structures (BHT and TD-HOFT) greatly reduce the computational overhead for both GC and group members from linear complexity to logarithmic complexity. Considering a member's computational overhead in join rekeying, protocols based on dynamic structures (D-HOFC and TD-HOFT) are less efficient than those based on corresponding static counterpart (DHC and BHT) because the former require member to update their personal key whereas the latter do not. Compared with protocol based on the static tree structure (BHT), protocols based on the dynamic counterpart (TD-HOFT) have a scalable storage requirement for a group member, and thus make more efficient use of storage resource. The EKT+ protocol is as efficient as the EKT protocol in leave rekeying, but outperforms the latter in join rekeying. According to Table 2, all user-based MKD protocols have a constant multicast size.

As discussed in Section 4.3.3, Protocol II, the EKT protocol and the EKT+ protocol are more suitable for constructing a hybrid protocol with tunable collusion-bandwidth tradeoffs than the LORE protocol and the LW protocol. Unlike the protocols based on static group access structures (the LORE protocol and the LW protocol), a particular type of non-members collusion attack (i.e., a coalition between a former evictee and a later-joining member) is useless for protocols based on dynamic group access structures (Protocol II, the EKT and EKT+ protocols).

## 6.3 Speed Benchmark Test with Related Cryptographic Primitives

TABLE 3
SPEED BENCHMARK

| Algorithm | *Input* | *Avg. Exe. Time* (μs) |
|---|---|---|
| **SHA-1** | 160-bit RND | 0.692 |
| **SHA-256** | 256-bit RND | 1.268 |
| **SHA-512** | 512-bit RND | 2.161 |
| **Mod. Mul.** | two 1024-bit RNDs | 2.72 |
| **Mod. Sqr.** | 1024-bit RND | 2.772 |

*We use a 1024-bit Blum Integer as the modulus for both a modular multiplication operation and a Blum-William function (i.e., a modular square operation).*

If we choose a Blum-William function as the HOWP when implementing a D-HOFC or TD-HOFT, our proposed protocols will involve intensive computations of modular multiplications (see Tables 1 and 2). To evaluate the performance of a modular multiplication operation, we utilize a popular software library MIRACL (version 5.5.4) [36] to carry out a speed benchmark test with related primitives including SHA-1 [37], SHA-256, SHA-512, modular multiplication, and modular square (Blum-William function). All five primitives have been implemented in C in MIRACL 5.5.4. The hardware and softwares on which this benchmark test was conducted are a notebook with Intel core i5-2410M 2.3 GHZ CPU and 4G memory, Ubuntu 11.10 64-bit operating system, and GCC 4.6.1 compiler. For every primitive, we calculate its average execution time over 1000 runs. The results provided in Table 3 show that SHA-1 is fastest of all five algorithms. However, recent cryptanalytic advances [38],[39] suggested that it is no longer safe to use either MD5 [40] or SHA-1. The UN National Institute of Standards and Technology (NIST) recommends that people should use extended block-size versions SHA, such as SHA-256 or SHA-512. According to Table 3, a modular multiplication operation is over 2 times slower than SHA-256. To make a fair comparison, we assume that a Blum-William function with a 1024-bit modulus is chosen as the HOWP for both D-HOFC and TD-HOFT, and SHA-256 as the one-way hash function for both DHC and BHT in the rest of this paper.

## 6.4 Experiment Using MBone User Activity Data

Although static group size $N$ is far bigger than dynamic group size $n$ for large dynamic groups in most time of a session, the computational cost of a modular multiplication $C_M$ is over twice bigger than that of one hash $C_h$ (SHA-256) according to Table 3. To compare the computational performance of a static structure based protocol (determined by both $N$ and $C_h$) with that of the corresponding dynamic structure based protocol (determined both by $n$ and $C_M$), we have to simulate how group size changes with time in reality and investigate how each protocol's performance changes accordingly.

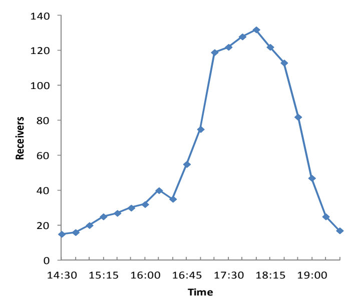


Fig. 7 UCB Seminar temporal statistics

Almeroth and ımmar [41],[42] eveloped a tool alled Mlisten that an collect the ɔin/leave times for ıulticast group ıembers in MBone udio sessions. Our xperiment is based n group dynamics ata collected by ..em for a UCB Multimedia Lecture Series audio session on February 17, 1995, shown in Figure 7. Our experimental goal is to investigate how performance changes with time in practice for every related user-based MKD protocol, and thus compare computational performance of a protocol based on dynamic group access structure with that of the corresponding protocol based on the static counterpart.

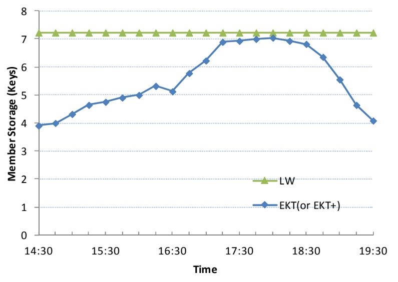


Fig. 8 Storage requirement for a member

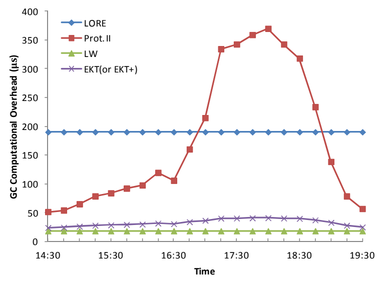


Fig. 9 GC computational overhead in join rekeying

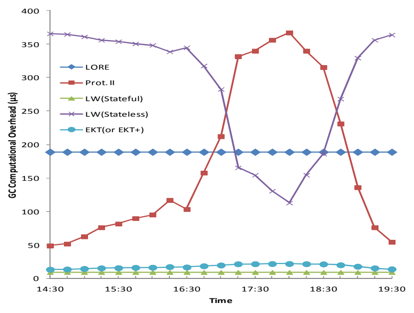


Fig. 10 GC computational overhead in leave rekeying

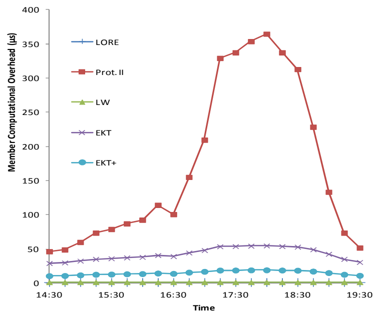


Fig. 11 Member computational overhead in join rekeying

From the experimental results shown in Figures 8-12, we can see that except for the stateless leave rekeying algorithm of the LW protocol, the storage and computational overhead of protocols based on static group access structures (DHC or BHT) — the LORE protocol and the LW protocol — remains constant. Whereas the computational overhead of protocols based on dynamic group access structures (D-HOFC or TD-HOFT) changes with time. For Protocol II that is based on a dynamic chain structure — D-HOFC, the shape of computational over-

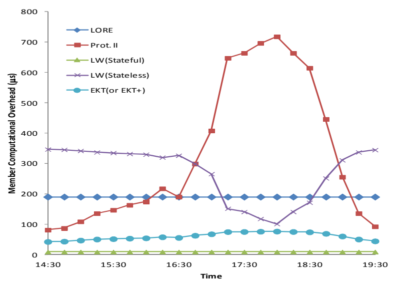


Fig. 12 Member computational overhead in leave rekeying

head trace (shown in Figures 9-12) matches that of group size trace (shown in Figure 7) perfectly, because the computational overhead is a linear function of group size $n$ (see Table 2). For the EKT and EKT+ protocols that are based on a dynamic tree structure — TD-HOFT, the trace of computational overhead is much smoother than that of group size, because the computational overhead is a logarithmic function of group size $n$ (see Table 2). For the stateless leave rekeying algorithm of the LW protocol, the shape of computational overhead trace is like that of a trace obtained by turning the group size trace upside down, because the computational overhead is a nearly-linear function of $r$ ($r$=$N$-$n$). Figure 8 confirms that the EKT and EKT+ protocols have a scalable storage requirement for a member compared with the LW protocol. According to the results shown in Figures 9 and 10, for both join rekeying and leave rekeying, GC's computational overhead of Protocol II is much less than that of the LORE protocol in almost 60% time of a MBone audio session. For leave rekeying, member's computational overhead of Protocol II is less than that of the LORE protocol in almost 40% time of a MBone audio session. Figure 11 confirms that for join rekeying, a member's computational overhead of the EKT+ protocol is much lower than that of the EKT protocol. From Figures 9-12, we can see that protocols based on a tree structure (BHT or TD-HOFT) outperform those based on a chain structure (DHC or D-HOFC). Compared with the protocol based on BHT (the LW protocol), protocols based on TD-HOFT (the EKT and EKT+ protocols) are more collusion-resistant and have a scalable storage overhead while maintaining computational efficiency comparable to the former.

## 7 Conclusion and Future Research

For the first time, we introduced the concept of group access structure for MKD protocols and provided a fresh view on the modular design of MKD protocols which is centered on group access structures. This view helps us gain an insight into MKD protocol design.

We introduced two new dynamic HOWP-based group access structures — D-HOFC and TD-HOFT. We then proposed three protocols — Protocol I, Protocol II (both based on D-HOFC), and EKT+ (based on TD-HOFT). By experimental results and comprehensive comparisons, we demonstrated that our protocols based on these dynamic group access structures have their own advantages compared with those based on the corresponding static counterparts (DHC and BHT).

So far, we have introduced three types of HOWP-based dynamic group access structures (HOFT [25], D-HOFC and TD-HOFT), all of which have found meaningful applications in designing MKD protocols. Compared to their hash-based counterparts (bottom-up OFT [1], DHC, and BHT), these HOWP-based structures can be not only derived as efficiently as the former (if we select a Blum-William function as the HOWP), but also updated without rekeying the entire group unlike the former. It would be interesting to find further applications for these HOWP-based structures.

## Acknowledgment

The authors would like to thank the anonymous referees for their helpful comments about this work. The work of J. Liu and Y. Zhang was supported by the Faculty Team Construction Foundation of Yunnan University under Grant XT412001. The work of Q. Huang was supported in part by the National Natural Science Foundation of China under Grant 61103232. The work of B. Yang was supported in part by the National Natural Science Foundation of China under Grants 60973134 and 61173164.

## References

[1] A. T. Sherman, and D. A. McGrew, “Key establishment in large dynamic groups using one-way function trees,” *IEEE Trans. Software Engineering,* vol. 29, no. 5, pp. 444-458, 2003.

[2] M. Steiner, G. Tsudik, and M. Waidner, “Key agreement in dynamic peer groups,” *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 8, pp. 769-780, 2000.

[3] Y. Kim, A. Perrig, and G. Tsudik, “Tree-based group key agreement,” *ACM Trans. Information and System Security,* vol. 7, no. 1, pp. 60-96, 2004.

[4] S. Rafaeli, and D. Hutchison, “A survey of key management for secure group communication,” *ACM Computing Surveys,* vol. 35, no. 3, pp. 309-329, 2003.

[5] Y. Challal, and H. Seba, “Group key management protocols: a novel taxonomy,” *International Journal of Information Technology,* vol. 2, no. 2, pp. 105-118, 2005.

[6] S. Zhu, and S. Jajodia, "Scalable group key management for secure multicast: a taxonomy and new directions," *Network Security*, S. C. H. Huang, D. MacCallum and D.-Z. Du, Eds., pp. 57-75, Springer US, 2010.

[7] C. K. Wong, M. Gouda, and S. S. Lam, “Secure group communications using key graphs,” *IEEE-ACM Trans. Networking,* vol. 8, no. 1, pp. 16-30, 2000.

[8] D. M. Wallner, E. J. Harder, and R. C. Agee, "Key management for multicast: issues and architectures," *Internet Draft*, Internet Eng. Task Force, 1998.

[9] G. Caronni, K. Waldvogel, D. Sun, and B. Plattner, “Efficient security for large and dynamic multicast groups,” *Proc. 7th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 376-383, 1998.

[10] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, “Multicast security: a taxonomy and some efficient constructions,” *Proc. IEEE INFOCOM*, pp. 708-716, 1999

[11] A. Perrig, D. Song, and D. Tygar, “ELK, a new protocol for efficient large-group key distribution,” *Proc. IEEE Symposium on Security and Privacy*, pp. 247-262, 2001.

[12] M. Waldvogel, G. Caronni, S. Dan, N. Weiler, and B. Plattner, “The VersaKey framework: versatile group key management,” *IEEE J. Selected Areas in Communications,* vol. 17, no. 9, pp. 1614-1631, 1999.

[13] D. Naor, M. Naor, and J. B. Lotspiech, "Revocation and tracing schemes for stateless receivers," *Proc. Advances in Cryptology*, pp. 41–62, 2001.

[14] D. Halevy, and A. Shamir, "The LSD broadcast encryption scheme," *Proc. Advances in Cryptology*, pp. 47-60, 2002.

[15] R. S. Douglas, *Cryptography Thoery and Practice*, Third ed., CRC Press, 2005.

[16] L. Cheung, J. A. Cooley, R. Khazan, and C. Newport, “Collusion-resistant group key management using attribute-based encryption,” *Cryptology ePrint Archive Report 2007/161*, 2007.

[17] Z. Zhou, and D. Huang, “On efficient ciphertext-policy attribute based encryption and broadcast encryption,” *Proc. 17th ACM conference on Computer and communications security*, pp. 753-755, 2010.

[18] S. Berkovits, "How to broadcast a secret," *Proc. Advances in Cryptology*, pp. 535-541, 1991.

[19] M. Naor, and B. Pinkas, "Efficient trace and revoke schemes," *Financial Cryptography*, Y. Frankel, Ed., pp. 1-20, 2001.

[20] L. Harn, and L. Changlu, “Authenticated group key transfer protocol based on secret sharing,” *IEEE Trans. Computers,* vol. 59, no. 6, pp. 842-846, 2010.

[21] G. H. Chiou, and W. T. Chen, “Secure broadcasting using the secure lock,” *IEEE Trans. Software Engineering,* vol. 15, no. 8, pp. 929-934, 1989.

[22] D. Micciancio, and S. Panjwani, "Corrupting one vs. corrupting many: The case of broadcast and multicast encryption," *Proc. International Colloquium on Automata, Languages and Programming*, pp. 70-82, 2006.

[23] J. Liu, and C. J. Wang, “Exclusive key based group rekeying,” *Cryptology ePrint Archive, Report 2011/575*, 2011.

[24] B. Briscoe, “MARKS: zero side effect multicast key management using arbitrarily revealed key sequences,” *Proc. Networked Group Communication*, pp. 301-320, 1999.

[25] J. Liu, and B. Yang, “Collusion-resistant multicast key distribution based on homomorphic one-way function trees,” *IEEE Trans. Information Forensics and Security,* vol. 6, no. 3, pp. 980-991, 2011.

[26] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha, “Key management for secure lnternet multicast using Boolean function minimization techniques,” *Proc. IEEE INFOCOM*, pp. 689-698, 1999.

[27] Z. Zhou, and D. Huang, “An optimal key distribution scheme for secure multicast group communication,” *Proc. IEEE INFOCOM*, pp. 1-5, 2010.

[28] J. Fan, P. Judge, and M. H. Ammar, “HySOR: group key management with collusion-scalability tradeoffs using a hybrid structuring of receivers,” *Proc. 11th International Conference on Computer Communications and Networks*, pp. 196 - 201, 2002.

[29] H. Kim, S. M. Hong, H. Yoon, and J. W. Cho, “Secure group communication with multiplicative one-way functions,” *Proc. International Conference on Information Technology: Coding and Computing*, Vol 1, pp. 685-690, 2005.

[30] B. Briscoe, and I. Fairman, "Nark: receiver-based multicast non-repudiation and key management," *Proc. 1st ACM conference on Electronic commerce*, pp. 22-30 , 1999.

[31] A. Fiat, and M. Naor, "Broadcast encryption," *Proc. Advances in Cryptology*, pp. 480-490, 1994.

[32] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," Research Report, Cambridge: Massachusetts Institute of Technology, Laboratory for Computer Science, 1979.

[33] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM,* vol. 21, no. 2, pp. 120-126, 1978.

[34] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean, "Self-healing key distribution with revocation," *Proc. IEEE Symposium on Security and Privacy*, pp. 241-257, 2002.

[35] S. Panjwani, "Private group communication: two perspectives and a unifying solution," PhD thesis, Computer Science and Engineering Department, University of California, San Diego, 2007.

[36] S. S. Ltd, "MIRACL (Multiprecision Integer and Rational Arithmetic C/C++ Library),", http://www.shamus.ie/, Last accessed on Nov., 2011.

[37] NIST, "Secure hash standard," *Federal Information Processing Standard*, FIPS-180-1, April 1995.

[38] X. Y. Wang, and H. B. Yu, "How to break MD5 and other hash functions," *Proc. Advances in Cryptology*, pp. 19-35, 2005.

[39] X. Y. Wang, Y. L. Yin, and H. B. Yu, "Finding collisions in the full SHA-1," *Proc. Advances in Cryptology*, pp. 17-36, 2005.

[40] R. Rivest, "The MD5 message-digest algorithm," *RFC 1321*, April 1992.

[41] K. C. Almeroth, and M. H. Ammar, "Collecting and modeling the join/leave behavior of multicast group members in the MBone," *Proc. 5th IEEE International Symposium on High Performance Distributed Computing*, pp. 209-216 , 1996.

[42] K. C. Almeroth, and M. H. Ammar, "Multicast group behavior in the Internet's multicast backbone (MBone)," *IEEE Communications Magazine,* vol. 35, no. 6, pp. 124-129, 1997.

**Jing Liu** received the PhD degree in computer application technology from the University of Electronic Science and Technology of China in 2003. From September 2003 to July 2005, he was with No. 30 Institute of China Electronics Technology Group Corporation as a postdoctoral fellow. From September 2005 to December 2012, he had been an assistant professor at the School of Information Science and Technology, Sun Yat-Sen University. Since January 2013, he has been with Yunnan University. His current research interests include applied cryptography and network security.He is a member of the IEEE.

**Qiong Huang** received his B.S. degree and M.S. degree from Fudan University in 2003 and 2006, respectively, and obtained Ph.D. degree from City University of Hong Kong in 2010. After graduation, he worked as a Research Fellow at Department of Computer Science, City University of Hong Kong. Now he is with South China Agricultural University. His research interests include cryptography and information security.

**Bo Yang** received the B. S. degree from Peking University in 1986, and the M. S. and Ph. D. degrees from Xidian University in 1993 and 1999, respectively. From July 1986 to July 2005，he had been at Xidian University, from 2002, he had been a professor of National Key Lab. of ISN in Xidian University, supervisor of Ph.D. He has served as a Program Chair for the fourth China Conference on Information and Communications Security (CCICS'2005) in May 2005, vice-chair for ChinaCrypt'2009 in Nov. 2009, and general chair for the Fifth Joint Workshop on Information Security (JWIS 2010), in Aug. 2010. He is currently professor and supervisor of Ph.D. at School of Computer Science, Shaanxi Normal University, a special-term professor of Shaanxi Province. His research interests include information theory and cryptography.

**Yang Zhang** received his B.S. Degree from Sun Yat-Sen University in 2011. He is currently pursuing his M.S. degree in computer science at Sun Yat-Sen University under supervision of Lecturer Jing Liu. His research interests include applied cryptography and network security.