Exclusive Key Based Group Rekeying Protocols

Jing Liu and Changji Wang

Abstract—In this paper, we first clarify the meaning of research on 1-resilient group rekeying protocols by showing that they are actually building blocks for constructing hybrid group rekeying protocols with tunable collusion-bandwidth tradeoffs. We then construct secure and efficient 1-resilient group rekeying protocols based on the idea of *exclusive key*. Given a group of users, an *exclusive key* for a user *i* is a key shared by all users in this group except *i*, and thus can be used to exclude *i* from this group effectively. We first present three personal key assignment algorithms based on this idea. The first is based on independent exclusive keys, and thus has a great storage requirement. The other two are based on functionally-dependent exclusive keys, and thus greatly reduce the storage requirement. Employing each personal key assignment algorithm, we propose both a stateful group rekeying protocol and a stateless one. We prove that all six protocols are secure against single-user attacks (i.e., 1-resilient) in a symbolic security model. Performance comparisons between our protocols and related ones show that either of the proposed Protocol III and Protocol III' is the best in its own class.

Index Terms—multicast key distribution, group rekeying, 1-resilient, collusion attack

1 Introduction

1.1 A Brief Survey of Research on Group Rekeying

Recent years have seen the rise of a large variety of group-oriented applications, for instance, pay-per-view, Pay-TV, DVB (Digital Video Broadcast), audio/video conferences, collaborative applications, stock quote streaming and so on. For some group-oriented applications like stock quote streaming, providing a security guarantee of data authenticity will suffice. However, for the other applications like pay-per-view, Pay-TV, audio/video conferences, providers would like to limit content distribution to subscribers who paid for the service. Thus, providing a security guarantee of confidentiality for group communication is mandatory. One of the most efficient ways to achieve private group communication is to use symmetric-key encryption. The group controller (GC) provides a common symmetric key called *group key* to all members of a group. Then a sender (or content provider) uses this shared group key to encrypt all digital contents and sends the encrypted data to all members over a broadcast channel. All members can use this shared group key to decrypt the encrypted data. However, the group key must be changed for every membership change due to the following reasons. To prevent a new member from decoding messages exchanged before it joined a group (if it has recorded earlier messages encrypted with the old group key), a new group key must be distributed for the group when a new member joins. This

Manuscript received 2011. Supported by the National Natural Science Foundation of China (No. 61173189)

- *Jing Liu is with the School of Information Science, and Technology and Guangdong Key Lab of Information Security and Technology, Sun Yat-Sen University, Guangzhou, 510006, China (e-mail: liujing3@mail.sysu.edu.cn).*
- *Changji Wang is with the School of Information Science, and Technology and Guangdong Key Lab of Information Security and Technology, Sun Yat-Sen University, Guangzhou, 510006, China (e-mail: isswchj@mail.sysu.edu.cn).*

security requirement is called *group backward secrecy*. On the other hand, to prevent a departing member from continuing access to the group's communication (if it keeps receiving the messages), the group key must be changed as soon as a member leaves. This requirement is called *group forward secrecy*. The process of updating the group key among all members upon every single change in group membership is referred to as (*immediate*) *group rekeying* in the literature. We further refer to group rekeying process induced by a member's departure (resp. join) as *leave rekeying* (resp. *join rekeying*). As we can expect, designing a scalable and efficient immediate group rekeying protocol for large dynamic multicast groups is a challenging problem.

Group rekeying protocols can be subdivided into two categories: the *stateful* and the *stateless*. For stateful group rekeying protocols, receivers must remain online and keep updating their internal states (personal keys). A successful decipher of current group key depends on successfully receiving all (or part) of past rekey messages. 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 successfully decipher any future rekey messages. On the contrary, for stateless group rekeying protocols, receivers are not allowed to maintain any internal state. After personal keys are given to registered receivers, they remain unchanged thereafter. The same setting is also assumed by broadcast encryption (BE) protocols [1], therefore broadcast encryption protocols can also be regarded as stateless group rekeying protocols. For example, the famous subset difference based member revocation (SDR) protocol [2] is regarded both as a BE protocol and as a stateless group rekeying protocol in the literature. In these protocols, current rekey message is independent of past rekey messages. A legitimate member can extract the group key from current rekey message even if it has missed all previous rekey messages. Thus, these protocols are very well suitable for application scenarios where members may go off-line frequently (e.g., pay-TV systems), or communication channel is lossy, or no feedback channel exists (e.g., encrypted DVD distribution).

The last decade has seen an exciting growth of research on group rekeying protocols (See [3] for an excellent survey, and two recent surveys are available in [4] and [5]). Most of group rekeying protocols in the literature are stateful. Among all stateful group rekeying protocols, the tree-based ones are most efficient and popular. The first tree-based stateful group rekeying protocol called *Logical Key Hierarchies* (LKH) was independently suggested by Wong *et al.* [6], Wallner *et al.* [7], and Caronni et al.[8]. Since the LKH protocol, various stateful group rekeying protocols [9],[10],[11],[12] based on logic key tree were proposed. For a multicast group of n users, immediate stateful group rekeying following these protocols has communication, computational and storage complexity all logarithmic in n. In highly dynamic groups, immediate rekeying upon every single membership change may be too expensive. To cope with this problem, researchers proposed to use *batch* or *periodic rekeying* [13], [14] instead. In *periodic rekeying*, a group rekeying operation will not be initiated until a predetermined number of time units have passed irrespective of membership dynamics. In *batch rekeying*, a group rekeying operation will not be initiated until a predetermined number of group membership changes have happened. Obviously, batch group rekeying is more general than immediate group rekeying. Given a batch group rekeying protocol, we immediately obtain a corresponding immediate group rekeying one by simply restricting the number of changes in group membership to 1.

Typical stateless group rekeying (BE) protocols are those based on so-called subset cover framework [2]: the complete sub-tree (CS) protocol [2], the subset difference (SD) protocol [2], and the layered subset difference(LSD) protocol [15]. Removing R members from the group of size n following the CS protocol requires transmission of $O(R\log_2(n/R))$ encrypted keys. Each member needs to store $O(\log_2 n)$ keys and perform 1 decryption operation. Revocation of R members following the SD protocol requires transmission of at most $2R$-1 encrypted keys. Each user needs to store $O(\log_2{}^2 n)$ keys and perform $O(\log_2 n)$ computations. Revocation of R members following the LSD protocol requires transmission of $O(R)$ encrypted keys. The protocol requires each user holds $O(\log_2{}^{3/2} n)$ keys and performs $O(\log_2 n)$ computations.

1.2 The Meaning of Research on 1-Resilient Group Rekeying Protocols

Recent research by Micciancio and Panjwani [16] confirmed that $O(\log_2 n)$ is actually the lower bound on the communication complexity of generic collusion-resistant group rekeying protocols. This lower bound reveals that it is impossible to achieve a lower communication overhead than $O(\log_2 n)$ without trading off some degree of collusion resistance. Fiat and Naor [1] introduced the following concept called *k-resilience* to quantify the degree of collusion resistance. Denote by U={1, 2,…, n} the universe of all users. A group rekeying protocol is called *resilient* to a set $S \subseteq U$, if for every subset $T \subseteq U \backslash S$, no eavesdropper that has all secrets associated with members of S, can obtain "knowledge" of the secret common to T. A protocol is called *k-resilient* if it is resilient to any set $S \subset U$ of size k. According to this definition, a protocol is called *1-resilient* if it is secure against any single-user attack, but a coalition of two receivers might break its security.

As claimed by Fan et al. in [17], some commercial content delivery applications (e.g., Internet based pay-per-minute audio/video broadcast program provider) will be extremely cost sensitive and willing to trade off some degree of collusion resistance for a decrease in rekeying message cost. Some resource-constraint environments may also require a communication complexity lower than $O(\log_2 n)$. Interestingly, Fan et al. [17] proposed a framework called a hybrid structure of receivers (HySOR) which is tunable between the LKH protocol [6], [7], [8] and a 1-resilient protocol called the LORE protocol [17] (see Section 2.2 for details) extremes in the sense that one can trade off some degree of collusion resistance for a decrease in broadcast size. Enlightened by this research, we will show in Section 3 that any 1-resilient group rekeying protocol (stateful or stateless) with constant communication overhead can be used in conjunction with a collusion-resistant tree-based protocol to construct a hybrid protocol with tunable collusion-bandwidth tradeoffs. On the other hand, Fiat and Naor [1] also showed that *k*-resilient BE (stateless group rekeying) protocols can be constructed from 1-resilient BE protocols. Therefore, research on designing secure and efficient 1-resilient group rekeying protocols has its own value despite the fact that they are only secure against single-user attack.

1.3 Contributions and Organization

Besides clarifying the meaning of research on 1-resilient group rekeying protocols, we construct 1-resilient protocols based on a useful concept called *exclusive keys*. Given a group of n users, an *exclusive key* for a user i is a key shared by all users in this group except i, and thus can be used to exclude i from the group. In this paper, we present three personal key assignment algorithms based on this idea. The first algorithm is based on independent exclusive keys, and has a storage requirement of $O(n)$ keys. The second is based on functionally-dependent exclusive keys which are derived using a *dual hash chain*, and thus greatly reduce the storage requirement from $O(n)$ keys to $O(1)$ keys but at the cost of an average computational complexity of $O(n)$ evaluations of a one-way hash function. The third is also based on functionally-dependent exclusive keys which are derived using a *binary hash tree*, and has a storage requirement of $O(\log_2 n)$ keys and an average computational complexity of $O(\log_2 n)$ evaluations of a one-way hash function. Employing each of these algorithms, we propose both a stateful group rekeying protocol and a stateless group rekeying protocol. Among the proposed six protocols, the proposed Protocol III is in fact an improved version of a famous stateless group rekeying protocol proposed by Fiat and Naor [1]. We use one of the subset-cover techniques called the *complete sub-tree method* [2] to improve it with respect to both computational overhead and security. We prove that all six protocols are secure against single-user attacks (i.e., 1-resilient) in a symbolic security model. Performance comparisons between our protocols and related ones show that either of the proposed Protocol III and Protocol III′ is the best in its own class.

The rest of this paper is organized as follows. Section 2 reviews some related research results on 1-resilient group rekeying protocols. In Section 3, we show that any 1-resilient group rekeying protocol (stateful or stateless) with constant communication overhead can be used in conjunction with a collusion-resistant tree-based protocol to construct a hybrid protocol with tunable collusion-bandwidth tradeoffs. In Section 4, we present formal definitions of exclusive keys and three personal key assignment algorithms respectively based on *independent exclusive keys*, a *dual hash chain* and a *binary hash tree*. Section 5 presents six group rekeying protocols respectively based on these three algorithms. In Section 6, we prove all proposed protocols are secure against single-user attacks in a symbolic security model. In section 7, we give performance comparisons between our protocols and existing 1-resilient group rekeying protocols. Section 8 concludes this paper and gives some interesting topics for future research.

2 Related Research on 1-Resilient Group Rekeying

Recall that we denote by $U=\{1, 2,\ldots, n\}$ the universe of all users. In the rest of our paper, we 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 $R^{(t)}$ (note that $R^{(t)}$ actually represents a subset that includes all identities of receivers that have been revoked up to time t and all identities that have not been assigned yet up to time t). For an arbitrary set S, we use $|S|$ to denote the order of S. The group key used to encrypt all information sent to $S^{(t)}$ is denoted by $GK^{(t)}$. Below, we interchangeably refer to $S^{(t)}$ as the group of legitimate users at time t or the set of *ids* of users in $S^{(t)}$. We use $\{M\}_K$ to denote the symmetric encryption of M by K.

2.1 1-Resilient Stateless Group Rekeying (BE) Protocols

In [1], Fiat and Naor proposed a *1-resilient* broadcast encryption (BE) protocol based on pseudo-random functions. A complete binary tree is derived from a random seed by repeatedly applying two pseudo-random functions in a top-down manner as follows. Suppose that the two pseudo-random functions are denoted by f_1 and f_2 respectively, and for an arbitrary intermediate node S, its left child is denoted by S_l and its right child by S_r. We have $S_l=f_1(S)$ and $S_r=f_2(S)$. Every user is associated with one of the leaves of the derived binary tree. Assignment of personal keys to users is based on the derived binary tree such that each leaf key is known to all users except its associated user. To revoke multiple members, GC only needs to Xor all their associated leaf-keys and uses the resulting key to encrypt the new group key. Thus all members except those evictees can extract the new group key from the rekey message. For convenience, we call this protocol the FN 1st protocol in the rest of this paper.

Fiat and Naor [1] also proposed a 1-resilient BE protocol that is cryptographically equivalent to the RSA scheme [18]. GC chooses a random RSA composite $N=P*Q$ where P and Q are distinct primes. Given the group Z_N^*, GC selects an element g with high order and keeps g secret. User i is assigned personal key $g_i = g^{p_i} \bmod N$, where p_i and p_j are relatively prime for all i, $j\in U$. All users know what user index refers to what p_i. For a privileged subset of users $S^{(t)}$, GC computes the corresponding group key $GK^{(t)}$ as $GK^{(t)} = g^{\prod_{i\in S^{(t)}} p_i} \bmod N$ and broadcasts $S^{(t)}$ (or the corresponding revocation list $R^{(t)}$). After receiving the list $S^{(t)}$ or $R^{(t)}$, every user i in $S^{(t)}$ can compute $GK^{(t)} = g_i^{\prod_{j\in S^{(t)}\setminus\{i\}} p_j} \bmod N$. Fiat and Naor proves that if the problem of *root extraction modulo an RSA composite* is hard, then this protocol is secure against single-user attacks (1-resilient). It is readily seen that any two user can collude to compute g. For convenience, we call this protocol the FN 2nd protocol in the rest of this paper.

2.2 1-Resilient Stateful Group Rekeying Protocols

Stateful (immediate or batch) join rekeying can be conducted trivially in either of the following two manners:

1) When a few new members join at time t, GC simply broadcasts a new group key $GK^{(t+1)}$ encrypted under current group key $GK^{(t)}$, i.e., the rekey message is $\left\{GK^{(t+1)}\right\}_{GK^{(t)}}$. Thus, all members except the joining members can decrypt $GK^{(t+1)}$ from the rekey message. For each of the joining members, GC sends it the new group key $GK^{(t+1)}$ and its personal keys over a secure unicast channel;

2) GC can first apply a public one-way hash function h (MD5 [19], or SHA-1[20]) to current group key GK to obtain the new group key as $GK^{(t+1)}=h(GK^{(t)})$, then broadcast a rekey notification message. After receiving this message, all members except the joining members can derive the group key $GK^{(t)}$ as above.

Compared with join rekeying, designing an efficient leave rekeying algorithm is much harder, because a revoked member always brings out information that may be used to encrypt future group keys. To ensure group forward secrecy, we must invalidate all knowledge held by an evictee. However, this is not a trivial task. The following 1-resilient group rekeying protocols all employ a join rekeying algorithm like either of the above. Thus, we only need to introduce their leave rekeying algorithms.

Fan et al. [17] proposed a 1-resilient stateful group rekeying protocol called *linear ordering of receivers* (*LORE*) which has constant communication overhead. Assignment of personal keys (also known as control keys in [17]) to users is based on two independent one-way hash chains, respectively called *forward chain* and *backward chain*. GC assigns each user two control keys, respectively selected from the forward chain and backward chain according to its rank in both chains. In leave rekeying, these two control keys in conjunction with current group key will be used to encrypt the new group key such that all users except the evictee can extract the new group key.

Kim et al. [21] proposed a 1-resilient stateful group rekeying protocol with constant communication overhead whose personal key assignment is based on a structure similar to a *binary hash tree* (BHT) (see Section 4.3 for details) except that instead of using traditional one-way hash functions to derive a binary key tree, they chose to use multiplicative one-way functions to facilitate updating the whole key tree when a member leaves or joins the group. For convenience, we call their protocol the KHYC protocol in the rest of this paper.

In [12], Waldvogel et al. proposed a 1-resilient stateful group rekeying protocol known as the *flat-table* (FT) *protocol* in the literature. Its personal key assignment is based on a simple structure called *flat-table*. The table contains one entry for the group key and $2log_2n$ more entries for key encryption keys (KEKs), where log_2n is the number of bits in the member *id* (because we need $\log_2 n$ bits to uniquely identify each member for a group of size n). There are two keys available for each bit in the member *id*, one associated with each possible value of the bit. A member knows only the key associated with the state of its bit. In total, each member holds log_2n+1 keys. Chang et al. [22] independently proposed a similar protocol and employed *Boolean function minimization techniques* (BFM) to greatly reduce the communication overhead of batch rekeying. However, compared with the LORE protocol and the KYHC protocol, immediate rekeying following these flat-table based protocols require transmission of $O(log_2n)$ encrypted keys.

3 From 1-Resilient Group Rekeying to That with Tunable Collusion-Bandwidth Tradeoffs

For all 1-resilient stateful/stateless group rekeying protocols reviewed in Section 2, a coalition of any two revoked receivers can break their group forward secrecy. Given a 1-resilient stateful (resp. stateless) group rekeying protocol with constant communication overhead, a straightforward solution to improve its collusion resistance is simply as follows. A group of n receivers are divided into d divisions. GC simply employs an independent instance of the 1-resilient stateful/stateless group rekeying protocol (i.e., each set of control keys respectively associated with each division is independent from each other) for each division to rekey group keys. Since the communication overhead of the 1-resilient stateful/stateless group rekeying protocol is $O(1)$ encrypted keys, the communication overhead of the straightforward solution is $O(d)$ encrypted keys. Now we analyze this solution's vulnerability to collusion attacks. To achieve the highest level of collusion protection, GC has to allocate a receiver to a randomly-chosen division when it joins the group. 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. If we use P_k to denote the probability that at least two of the k colluding receivers are allocated in the same division, then according to [17] we have

$$P_k = \begin{cases} 1, & k > d \\ 1 - \dfrac{d!(n-k)!(\frac{n}{d})^k}{n!(d-k)!}, & k \le d. \end{cases}$$

It is readily seen that for given n and k, P_k is a decreasing function with respect to d. In other words, to reduce vulnerability to collusion attack, we must increase the number of divisions, d. But a greater d means a higher communication overhead (which is proportionate to d). Therefore, there exists a tradeoff between collusion resistance and bandwidth.

However, an obvious drawback of the above solution is that its communication overhead is linearly proportional to the number of divisions. In the following, we will show how to further reduce the communication overhead of the straightforward solution for stateful group rekeying (resp. the straightforward solution for stateless group rekeying) while maintaining the same level of security. In the following, we call the above straightforward solution for stateful (resp.

stateless) group rekeying the *stateful straightforward solution* (resp. the *stateless straightforward solution*) for short.

3.1 From 1-Resilient Stateful Group Rekeying to That with Tunable Collusion-Bandwidth Tradeoffs

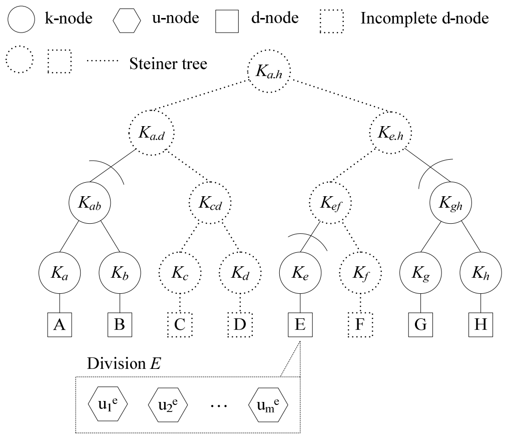

Fig.1. Hybrid structure

Fan et al. [17] proposed a framework called a hybrid structure of receivers (HySOR) which can be used to reduce the stateful straightforward solution's communication complexity from $O(d)$ to $O(log_2d)$ without compromising its security. HySOR uses a 1-resilient protocol — the LORE protocol (see Section 2.2 for some details) and the LKH protocol [6], [7], [8] to achieve tunable collusion-bandwidth tradeoffs. As illustrated in Figure 1, receivers are also divided into d *divisions* and each division is logically regarded as a leaf node (known as *d-node*) in an LKH key tree. Division keys are represented by the leaf *k*-nodes in an LKH key tree. Each receiver's personal keys include not only the control keys as required by the LORE protocol but also the auxiliary keys corresponding to the nodes in the path from its division key to the root as required by the LKH protocol. For each division, GC employs an independent instance of the LORE protocol to rekey the corresponding division key when there is a change in membership happened in that division. For simplicity, we assume that every division has at least one active receiver after a change in membership. When some division has no active receiver (i.e., it becomes *inactive*), the group rekeying procedure is a little different from the following. Please refer to [17] for details. When a receiver of the *i*-th division joins/leaves the group, GC first uses the *i*-th instance of the LORE protocol to update the corresponding division key K_i, and then uses the LKH protocol to update all the keys corresponding to the nodes in the path from K_i to the root of the key tree (note that the root node corresponds to the group key). Of course, the LORE protocol used by HySOR can be substituted by any 1-resilient stateful group rekeying protocol with constant communication overhead. Suppose that the constant communication overhead of the 1-resilient protocol is C, then the communication overhead of the resulting hybrid protocol is only $2\log_2 d+C$ encrypted keys. That of the LKH protocol is $2\log_2 n$ encrypted keys, where $n>>d$. Since the LKH protocol has been proved free from collusion attacks [23], the resulting hybrid protocol maintains the same level of security as the stateful straightforward solution. And there is also a trade-off between collusion resistance and bandwidth like the stateful straightforward solution.

3.2 From 1-Resilient Stateless Group Rekeying to That with Tunable Collusion-Bandwidth Tradeoffs

We can use a collusion-resistant tree-based stateless protocol (like those based on the *subset-cover framework*) to construct a hybrid stateless protocol which reduces the stateless straightforward solution's communication overhead without compromising its security. Since all schemes based on the subset cover framework — the complete sub-tree (CS) protocol [2], the subset difference (SD) protocol [2], the layered subset difference(LSD) protocol [15] — have been proved collusion-resistant, each of them can be used to construct this hybrid protocol. Personal key assignment is also based on the hybrid structure illustrated in Figure 1. Specifically, each receiver's personal keys include not only those control keys as required by a 1-resilient stateless protocol but also those auxiliary keys in the key tree as required by a subset-cover based stateless protocol. For example, for the CS protocol, the latter are those keys along the path from the corresponding d-node to the root just like the LKH protocol. We call a division and its corresponding d-node in the key tree *complete* if all receivers in that division are current members of a group; otherwise we call them *incomplete*. For each incomplete division, GC simply uses an independent instance of a 1-resilient stateless group rekeying protocol to rekey group keys. From perspective of the subset-cover based stateless group rekeying, all complete d-nodes are regarded as legitimate members while all incomplete ones as evictees. Following a subset-cover based stateless protocol, GC needs to first find those privileged subsets that together "cover" all complete d-nodes (we call these subsets *"covering" subsets* for short), and then use their corresponding auxiliary keys to encrypt the new group key. To exemplify this idea, we use the CS protocol [2] and an arbitrary 1-resilient stateless protocol to construct a hybrid protocol with tunable collusion-bandwidth tradeoffs. We reuse Figure 1 to explain how to perform group rekeying following this hybrid protocol when all evictees are distributed in divisions C, D and F (i.e., A, B, E, G and H are complete divisions). GC performs the following three steps to complete group rekeying: (1) For each incomplete division, GC simply uses an independent instance of the 1-resilient stateless protocol to rekey group keys; (2) GC uses the so-called *complete sub-tree method* on which the CS protocol is based [2] to find those "covering" subsets (that covers all complete divisions) and their corresponding auxiliary keys. Specifically, as illustrated in Figure 1, GC needs to first compute a Steiner tree that is spanning the root and all incomplete d-nodes (i.e., C, D, and F in this case). Then all auxiliary keys associated with those "covering" subsets are just those associated with the roots of those sub-trees that hang off the Steiner tree. Taking the case illustrated by Figure 1 as an example, these "covering" subsets are $\{A, B\}$, $\{E\}$, $\{G, H\}$, and their corresponding auxiliary keys are K_{ab}, K_e, and K_{gh}, respectively; (3) GC sends the new group key encrypted with these auxiliary keys respectively. For our example, GC sends the following rekey message: $\{GK'\}_{K_{ab}}, \{GK'\}_{K_e}, \{GK'\}_{K_{gh}}$.

Suppose that the group is divided into d divisions, and all m ($m>r$) evictees are uniformly distributed in r ($r<d$) incomplete divisions, then the communication overhead of this hybrid protocol is $O(r)+O(r\log_2(d/r))$ encrypted keys which is less than both that of the straightforward solution ($O(d)$) and that of the CS protocol ($O(m\log_2(n/m))$). Since the CS protocol has been proved free from collusion attacks [1], the hybrid protocol maintains the same level of security as the

stateless straightforward solution. There is a trade-off between collusion and bandwidth like the stateless straightforward solution.

In this section, we showed that any 1-resilient stateful (resp. stateless) group rekeying protocol with constant communication overhead can be used in conjunction with a collusion-resistant stateful (resp. stateless) tree-based protocol to construct a hybrid stateful (resp. stateless) protocol with tunable collusion-bandwidth tradeoffs. And communication overhead of the resulting hybrid protocol is exactly between that of 1-resilient protocol and that of the tree-based protocol. Therefore, research on designing secure and efficient 1-resilient group rekeying protocols has its own value despite the fact that they are only secure against single-user attack. In the following, we focus on designing efficient and secure 1-resilient group rekeying protocols with constant communication overhead.

4 1-Resilient Personal Key Assignment Algorithms Based on Exclusive keys

An informal concept of *exclusive key* was first given by Kim et al. [21] recently, while the idea actually originated in [1] more than a decade ago. In this section, we formalize this concept, and then propose three personal key assignment algorithms for group rekeying based on this idea. We also prove the security of them.

Definition 1 — Given a universe U of n users, an *exclusive key K_i for an arbitrary user $i \in U$* is a long-term key shared by all users in $U \backslash \{i\}$.

We can extend Definition 1 to obtain Definition 2.

Definition 2 — Given a universe U of n users, an *exclusive key K_S for an arbitrary subset $S \subseteq U$* is a long-term key shared by all users in $U \backslash S$.

In leave rekeying, an exclusive key K_i (resp. K_S) can be used by GC to exclude i (resp. all users in S) in the sense that GC can broadcast a new group key encrypted by K_i (resp. K_S) such that all users in U except i (resp. all users in S) can decrypt the rekey message.

Personal key assignment based on the idea of exclusive key is bound to be only 1-resilient, because a coalition of an arbitrary pair of users i and j can break their security as follows. According to Definition 1, i and j both hold an exclusive key that can be used to exclude each other. They can simply exchange those exclusive keys such that neither of them can be excluded by GC.

If we want to construct a feasible group rekeying protocol based on exclusive key, then its personal key assignment based on exclusive keys must at least satisfy the following three conditions:

1) *Correctness* — for an arbitrary exclusive key K_i for i, K_i must be shared by all users except i.

2) *Completeness* — for every user $i \in U$, i must hold or is able to derive the exclusive key K_j for all $j \neq i$. In fact, the key set $\{K_j \mid j \neq i\}$ or the intermediate seeds used to derive this key set will be assigned to i by GC as its personal key.

3) *Security* (1-resilience) — for every user $i \in U$, it is (information-theoretically) impossible or (computationally) infeasible for i to compute the exclusive key K_i for i given its personal key.

Now we present three personal key assignment algorithms that all fulfill the three conditions.

4.1 Personal Key Assignment Based on Independent Exclusive Keys (IEK) — the IEK Algorithm

TABLE 1
IEK ALGORITHM

$$\begin{array}{c} \textit{Ranks} \\ 1 \\ 2 \\ \vdots \\ i-1 \\ i \\ i+1 \\ \vdots \\ n \end{array} \quad \begin{array}{c} \textit{Exclusive Keys} \\ \begin{pmatrix} (\,) & K_2 & \cdots & K_{i-1} & K_i & K_{i+1} & \cdots & K_n \\ K_1 & (\,) & \cdots & K_{i-1} & K_i & K_{i+1} & \cdots & K_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ K_1 & K_2 & \cdots & (\,) & K_i & K_{i+1} & \cdots & K_n \\ K_1 & K_2 & \cdots & K_{i-1} & (\,) & K_{i+1} & \cdots & K_n \\ K_1 & K_2 & \cdots & K_{i-1} & K_i & (\,) & \cdots & K_n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ K_1 & K_2 & \cdots & K_{i-1} & K_i & K_{i+1} & \cdots & (\,) \end{pmatrix} \end{array}$$

GC (GC) first generates a set of n random (independent) keys denoted by $S=\{K_1,\ldots,K_n\}$. As illustrated in Table 1, each rank i (i=1,2,…,n) is associated with a set of keys $\{K_1,\ldots,K_{i-1}, K_{i+1},\ldots,K_n\}$ (i.e., $S \backslash \{K_i\}$). When user i joins the group, GC assign it a unique rank, namely i (for simplicity) and associated set of keys $S \backslash \{K_i\}$ as its personal key. In the following, we interchangeably refer to i as a rank or a user according to its context. Once user i is assigned a rank, i will occupy this rank and its associated personal key forever in the sense that they will never be assigned to other users even if i leaves the group. The effect of such a personal key assignment is such that each K_i is shared among all users except i, and can be used to exclude i in a leaving rekeying algorithm (correctness). For example, if GC wants to exclude user i, it broadcasts the new group key GK' encrypted by K_i such that all member except i can decrypt the new group key. It is easy to check that the IEK algorithm satisfies the completeness condition. According to the following theorem, this algorithm is information-theoretically secure against single-user attacks (1-resilience).

Theorem 1: *For an arbitrary user i, the exclusive key K_i for i is random given n-1 exclusive keys $\{K_1,\ldots,K_{i-1}, K_{i+1},\ldots,K_n\}$ (i.e., personal key of i).*

Proof: This claim is immediately followed from the fact that all n exclusive keys $K_1,\ldots,K_n$ are generated randomly and independently.

The storage requirement for each user is n-1 keys and that for GC is n keys. The computational cost for each user is zero.

4.2 Personal Key Assignment Based on Dual Hash Chain — the DHC Algorithm

This personal key assignment is similar to the IEK algorithm except that GC generates functionally-dependent exclusive keys instead of independent random ones as illustrated in Table 2. More specifically, GC first chooses two random seeds f_1 and b_1, and then generates two (n-1)-length one-way hash chains — forward chain and backward chain by repeatedly applying a public one-way hash function h to f_1 and b_1, respectively. That is, the (i+1)-th *forward key* f_{i+1} (resp. the (i+1)-th *backward key* b_{i+1}) is computed as $f_{i+1}=h(f_i)$ (resp. $b_{i+1}=h(b_i)$) (i=1,…,n-2). We call this structure comprised of two hash chains a

TABLE 2

DHC ALGORITHM

$$\begin{array}{cc}
\textit{Ranks} & \textit{Exclusive keys} \\
\begin{array}{c} 1 \\ 2 \\ \vdots \\ i-1 \\ i \\ i+1 \\ \vdots \\ n-1 \\ n \end{array} &
\left(\begin{array}{ccccccccccccc}
(\,) & f_1 \rightarrow & f_2 \rightarrow & \cdots & f_{i-3} \rightarrow & f_{i-2} \rightarrow & f_{i-1} \rightarrow & f_i \rightarrow & f_{i+1} \rightarrow & \cdots & f_{n-3} \rightarrow & f_{n-2} \rightarrow & f_{n-1} \\
b_{n-1} & (\,) & f_2 \rightarrow & \cdots & f_{i-3} \rightarrow & f_{i-2} \rightarrow & f_{i-1} \rightarrow & f_i \rightarrow & f_{i+1} \rightarrow & \cdots & f_{n-3} \rightarrow & f_{n-2} \rightarrow & f_{n-1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
b_{n-1} & \leftarrow b_{n-2} & \leftarrow b_{n-3} & \cdots & \leftarrow b_{n-i+2} & (\,) & f_{i-1} \rightarrow & f_i \rightarrow & f_{i+1} \rightarrow & \cdots & f_{n-3} \rightarrow & f_{n-2} \rightarrow & f_{n-1} \\
b_{n-1} & \leftarrow b_{n-2} & \leftarrow b_{n-3} & \cdots & \leftarrow b_{n-i+2} & \leftarrow b_{n-i+1} & (\,) & f_i \rightarrow & f_{i+1} \rightarrow & \cdots & f_{n-3} \rightarrow & f_{n-2} \rightarrow & f_{n-1} \\
b_{n-1} & \leftarrow b_{n-2} & \leftarrow b_{n-3} & \cdots & \leftarrow b_{n-i+2} & \leftarrow b_{n-i+1} & \leftarrow b_{n-i} & (\,) & f_{i+1} \rightarrow & \cdots & f_{n-3} \rightarrow & f_{n-2} & f_{n-1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
b_{n-1} & \leftarrow b_{n-2} & \leftarrow b_{n-3} & \cdots & \leftarrow b_{n-i+2} & \leftarrow b_{n-i+1} & \leftarrow b_{n-i} & \leftarrow b_{n-i-1} & \leftarrow b_{n-i-2} & \cdots & \leftarrow b_2 & (\,) & f_{n-1} \\
b_{n-1} & \leftarrow b_{n-2} & \leftarrow b_{n-3} & \cdots & \leftarrow b_{n-i+2} & \leftarrow b_{n-i+1} & \leftarrow b_{n-i} & \leftarrow b_{n-i-1} & \leftarrow b_{n-i-2} & \cdots & \leftarrow b_2 & \leftarrow b_1 & (\,)
\end{array}\right)
\end{array}$$

dual hash chain (DHC). For each rank i (i=1,2,…,n), GC associates i with two keys f_i and b_{n-i+1} (here we regard both b_n and f_n as null). With f_i and b_{n-i+1}, user i can derive all other exclusive keys as illustrated in Table 2 by repeatedly applying h. The effect of such a personal key assignment is that GC can exclude user i by broadcasting a rekey message as $\left\{\{GK'\}_{f_{i-1}}, \{GK'\}_{b_{n-i}}\right\}$. Thus users with ranks from 1 to i-1 can decrypt $\{GK'\}_{f_{i-1}}$ to obtain the new group key because all users with ranks from 1 to i-2 can derive the decryption key f_{i-1} and user i-1 holds f_{i-1} by itself. And users with ranks from i+1 to n can decrypt $\{GK'\}_{b_{n-i}}$ to obtain the new group key because all users with ranks from i+2 to n can derive the decryption key b_{n-i} and user i+1 holds b_{n-i} by itself. Whereas, user i can decrypt neither $\{GK'\}_{f_{i-1}}$ nor $\{GK'\}_{b_{n-i}}$. According to Definition 1, the exclusive key for user i is a pair of keys $\{f_{i-1}, b_{n-i}\}$. It is easy to check that the DHC algorithm satisfies the completeness condition. According to the following theorem, this algorithm is computationally secure against single-user attacks.

Theorem 2: *For an arbitrary user i, it is computationally infeasible to compute either of the exclusive key pair f_{i-1} and b_{n-i} given two exclusive keys f_i* and b_{n-i+1} (*i.e., personal key of i*).

Proof: This claim is immediately followed from the fact that h is a one-way hash function.

Compared with the IEK algorithm, the storage requirement for each user is only 2 keys at most, and for GC it is only two keys. However, to derive the pair of exclusive keys for an evictee, each user and GC need to perform $O(n)$ evaluations of a one-way hash function in average.

Remark 1: The personal key assignment algorithm of the LORE protocol [17] is also based on DHC, but is different from ours. Interestingly, both algorithms have the same storage and computation complexity.

4.3 Personal Key Assignment Based on Binary Hash Tree— the BHT Algorithm

As introduced in Section 2.1, to realize a 1-resilient broadcast encryption protocol — the FN 1st protocol, Fiat and Naor [1]

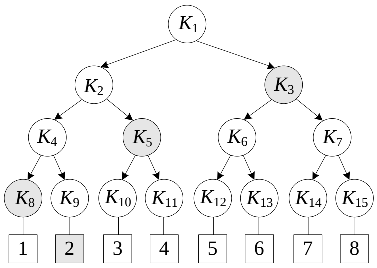

Fig. 2. Binary hash tree

proposed a personal key assignment algorithm based on a binary tree which is derived from a random seed by repeatedly applying two pseudo-random functions in a top-down manner. In practice, this binary tree can also be derived by using two computationally-efficient one-way hash functions, e.g., MD5 [19] or SHA-1 [20]. A binary tree derived by using hash functions is called *Binary Hash Tree* (BHT) by Briscoe [24], and he proposed a time-based multicast key distribution protocol based on BHT called MARKS [24]. Here, we use BHT for group rekeying protocols. GC first generates a *binary hash tree* (BHT) that has *n* leaf *k*-nodes as illustrated in Figure 2. Specifically, a BHT is derived from a random seed K_1 in a top-down manner such that given a *k*-node K_i, its left child K_{2i} is computed as $K_{2i} = f_L(K_i)$, and its right child K_{2i+1} as $K_{2i+1} = f_R(K_i)$ where f_L and f_R are two different one-way hash functions. Every user is associated with a unique leaf *k*-node. For convenience, given a user *i*, we call those keys in *i*'s path to the root *path keys* and those keys that are siblings of path keys *sibling keys*. When a user *i* joins the group, GC associates it with a unique leaf *k*-node and assigns it the corresponding sibling keys. With these sibling keys, *i* can compute all keys on the BHT except those path keys. For example, a user 2 associated with K_9 will be assigned a set of sibling keys $\{K_3, K_5, K_8\}$, and thus user 2 is able to compute all the keys on the BHT except the set of path keys $\{K_1, K_2, K_4, K_9\}$. The effect of such a personal key assignment is that an intermediate key K_i is shared among all users except those associated with the leaf nodes of the sub-tree rooted at K_i. Therefore K_i can be used to exclude all users respectively associated with the leaf *k*-nodes of a sub-tree rooted at K_i (see next section for details). According to Definition 2, K_i is the exclusive key for the set of users who are associated with the leaf nodes the sub-tree rooted at K_i (correctness). It is easy to check that the BHT algorithm satisfies the completeness condition. According to the following theorem, this algorithm is computationally secure against single-user attacks.

Theorem 3: *For an arbitrary user i, it is computationally infeasible to computer any of its path keys given its sibling keys.*

Proof: This claim is immediately followed from the fact that *h* is a one-way hash function.

The storage requirement for GC is only one key, but for each user it is $\log_2 n$ keys. However, to derive the exclusive key for an evictee, each user and GC only need to perform $O(log_2 n)$ evaluations of a one-way hash function in average.

***Remark 2*:** Let us consider the tradeoff between the two relevant resources at user end: storage overhead and computational overhead. The IEK-based algorithm achieves a storage complexity of $O(n)$ keys and zero computational cost at one end, whereas the DHC-based algorithm a storage complexity of $O(1)$ keys and an average computational complexity of $O(n)$ evaluations of one-way hash function at the other end. Between these two extremes, does the BHT algorithm achieve a storage complexity of $O(\log_2 n)$ keys and an average computational complexity of $O(\log_2 n)$ evaluations of one-way hash function.

5 Exclusive Key-Based 1-Resilient Group Rekeying Protocols

Using each of the three personal key assignment algorithms, we can design both a stateless group rekeying protocol and a stateful one. For the two protocols based on the same type of personal key assignment algorithm, we always introduce the stateless one before the stateful one because the batch leave rekeying algorithm of the latter may use the stateless protocol. Since immediate group rekeying is just a special case of batch group rekeying, we only need to deal with batch group rekeying. Below, we denote "multicast" by "⇒"and "secure unicast" by "→".

5.1 Group Rekeying Based the IEK Algorithm

5.1.1 A stateless group rekeying protocol — Protocol I

We first introduce the stateless group rekeying protocol (or BE protocol) that is based on the IEK algorithm — Protocol I. Whenever a new member joins the group, GC assigns it a free rank and sends it associated personal keys by a secure unicast channel. Suppose that there are changes in membership (joins or leaves) at time t and the privileged set changes from $S^{(t)}$ to $S^{(t+1)}$. Recall that the complement of $S^{(t+1)}$ — $R^{(t+1)}$ represents a subset that includes all ranks of users that have been revoked up to time t+1 and all ranks that have not been assigned yet up to time t+1. For the privileged set $S^{(t+1)}$, GC generates a random group key $GK^{(t+1)}$ and then sends the following message by multicast:

GC⇒$S^{(t+1)}$: $R^{(t+1)}, \left\{GK^{(t+1)}\right\}_{\oplus_{i\in R^{(t+1)}} K_i}$.

According to [1], every rekey message of a BE protocol (stateless group rekeying protocol) must contain a *set identification information* that is used to uniquely identify a privileged set. Since $|R^{(t+1)}|$ is usually far less than $|S^{(t+1)}|$, we choose to use the blacklist $R^{(t+1)}$ instead of the white list $S^{(t+1)}$ as the set identification information. Note that the key used to encrypt the group key is computed as $\oplus_{i\in R^{(t+1)}} K_i$, i.e., Xoring all the exclusive keys associated with all revoked and unassigned ranks in $R^{(t+1)}$.

The computational cost of GC is only 1 encryption and that of each group member is only one decryption. Note that we don't count the cost of the Xor operations in the remainder of this paper because the Xor operation is very computationally efficient. And the multicast size is only one encrypted key plus a blacklist that amounts to $|R^{(t+1)}|*\log_2 n$ bits (recall that it takes at least $\log_2 n$ bits to uniquely identify a member or a rank for a group of size n).

5.1.2 A stateful group rekeying protocol — Protocol I'

Let us introduce the stateful group rekeying protocol that is based on the IEK algorithm — Protocol I′. Suppose that m

users join the group $S^{(t)}\subset U$ at the same time t and we denote this set of users by $J=\{i_1, i_2,\dots, i_m\}$. For each $i_j\in J$ (j=1,…, m), GC needs to assign it a free rank also denoted by i_j. Then GC derives the new group key $GK^{(t+1)}$ by computing $GK^{(t+1)}=h(GK^{(t)})$ where h is a public one-way hash function, and sends the following messages:

GC⇒$S^{(t)}$: a *rekey notification*,

For each $i_j\in J$ (j=1,…, m)

GC→i_j: $\{K_1,\cdots,K_{i_j-1},K_{i_j+1},\cdots,K_n\}$, $GK^{(t+1)}$.

After receiving the rekey notification message, every current member in $S^{(t)}$ can derive the new group key $GK^{(t+1)}$ by computing $GK^{(t+1)}=h(GK^{(t)})$.

In fact, both immediate and batch leave rekeying can be supported by using the stateless counterpart — Protocol I in a straightforward way. Suppose that m users leave the group $S^{(t)}$ at the same time t and we denote this set of users by $L=\{i_1, i_2,\dots, i_m\}$. GC simply uses Protocol I to exclude all users in $R^{(t)}\cup L$ from obtaining the new group key. For convenience, we call this straightforward approach of using the stateless counterpart for batch leave rekeying *stateless leave rekeying* in the remainder of this paper. Here we introduce another approach called *stateful leave rekeying*. Because $GK^{(t)}$ is shared by all users in $S^{(t)}=U\setminus R^{(t)}$, it can be regarded as a temporary exclusive key for subset $R^{(t)}$ at time t, although it is not a long-term one as defined by Definition 2. Therefore, using the exclusive keys respectively for every currently-departing user in L in conjunction with $GK^{(t)}$ (i.e., Xoring them together) to encrypt the new group key $GK^{(t+1)}$ is enough to exclude all users in $R^{(t+1)}=L\cup R^{(t)}$ at time t. That is to say, GC needs to send the following rekey message:

GC⇒$S^{(t+1)}$: $i_1, i_2\cdots, i_m, \left\{GK^{(t+1)}\right\}_{\left(\oplus_{i_j\in L}K_{i_j}\right)\oplus GK^{(t)}}$ here $S^{(t+1)}=S^{(t)}-L$.

Every current member of $S^{(t)}$ except those in L can extract $GK^{(t+1)}$ from this message. Due to the usage of current group key, this type of batch leave rekeying is *stateful* (see Remark 3 at the end of this section for the reason).

For join rekeying, the computational cost of GC and each group member except the joining member is only one evaluation of a one-way hash function. The unicast size is $m*n$ keys in total and the multicast size is only one notification message. For leave rekeying, the computational cost of GC is only 1 encryption, and that of each group member is only 1 decryptions. And the multicast size is only 1 encrypted key plus $m*\log_2 n$ bits for identifying m departing members.

5.2 Group Rekeying Based on the DHC Algorithm

5.2.1 A stateless group rekeying protocol — Protocol II

In this section, we introduce the stateless group rekeying protocol that is based on the DHC algorithm — Protocol II. When a new member joins the group, GC must send him/her a corresponding personal key by unicast. Suppose that there are

TABLE 3

PROTOCOL II

$$\begin{array}{c c}
\textit{Ranks} & \textit{Exclusive keys} \\
\begin{array}{c} 1\rfloor \\ \times \\ \lceil 3 \\ 4\rfloor \\ \times \\ \lceil 6 \end{array} &
\begin{pmatrix}
(\ ) & f_1\rightarrow\rfloor & f_2\rightarrow & f_3\rightarrow & f_4\rightarrow & f_5 \\
b_5 & (\times) & f_2\rightarrow & f_3\rightarrow & f_4\rightarrow & f_5 \\
b_5 & \lceil\leftarrow b_4 & (\ ) & f_3\rightarrow & f_4\rightarrow & f_5 \\
b_5 & \leftarrow b_4 & \leftarrow b_3 & (\ ) & f_4\rightarrow\rfloor & f_5 \\
b_5 & \leftarrow b_4 & \leftarrow b_3 & \leftarrow b_2 & (\times) & f_5 \\
b_5 & \leftarrow b_4 & \leftarrow b_3 & \leftarrow b_2 & \lceil\leftarrow b_1 & (\ )
\end{pmatrix}
\end{array}$$

Note : We use '×' to locate revoked ranks.

changes in group membership (joins or leaves) at time t and the privileged set changes from $S^{(t)}$ to $S^{(t+1)}$. Given a privileged set $S^{(t+1)}$, ranks associated with all legitimate users (legitimate ranks for short) are distributed by the pattern of disjoint rank intervals. Or we would like to say legitimate ranks are covered by disjoint rank intervals. As illustrated in the first column of Table 3, if $U = \{1, 2,\ldots, 6\}$ and the privileged set of users is {1, 3, 4, 6}, legitimate ranks are covered by the following disjoint rank intervals: [1], [3,4] and [6]. We denote these rank intervals by $I_1, I_2,\ldots, I_r$. GC generates a random group key $GK^{(t+1)}$, and for each rank interval $I_j = [j_1, j_m]$ (i.e., rank interval I_j starts at rank j_1 and ends at rank j_m), creates a cipher text $C_{I_j} = \left\{GK^{(t+1)}\right\}_{b_{n-j_1+1}\oplus f_{j_m}}$. Referring to Table 3, it is readily seen that only members whose associated rank is covered by I_j can decrypt this cipher text according to the DHC algorithm. After creating all those cipher texts, GC sends the following rekey message by multicast:

GC⇒$S^{(t+1)}$: $I_1, I_2, \cdots, I_r, C_{I_1}, C_{I_2}, \cdots, C_{I_r}$.

We encourage the interested readers to refer to the proof of Theorem 7 in Section 6 for more details behind this construction. The number of disjoint rank intervals r is at most $|R^{(t+1)}|+1$. The computational cost of GC is $2n$ evaluations of one-way hash function (for computing the exclusive key pairs for all revoked members) and $|R^{(t+1)}|+1$ encryptions at most. The computational cost of each group member is n evaluations of one-way hash function at most plus one decryption. And the multicast size is $|R^{(t+1)}|+1$ encrypted keys plus $(|R^{(t+1)}|+1)*\ 2\log_2 n$ bits at most (recall that each rank interval is uniquely identified by two ranks, therefore $2\log_2 n$ bits).

5.2.2 A stateful group rekeying protocol — Protocol II'

Now we introduce the stateful group rekeying protocol that is based on the DHC algorithm — Protocol II′. Suppose that m users denoted by $J=\{i_1, i_2,\ldots, i_m\}$ join the group $S^{(t)}\subset U$ at the same time t. GD performs join rekeying similar to that of Protocol I′ except that the unicast message is as follows:

For each $i_j\in J$ (j=1,…, m)

GC→ i_j: $\left\{ f_{i_j}, b_{n-i_j+1} \right\}$, $GK^{(t+1)}$.

Both immediate and batch leave rekeying can be supported by using Protocol II in a straightforward manner. Suppose that m users leave the group $S^{(t)}$ at the same time t and we denote this set of users by $L=\{i_1, i_2,\ldots, i_m\}$. GC first computes all those rank intervals that cover all legitimate users in $S^{(t)}$-L and then uses Protocol II to exclude all users in $R^{(t)}\cup L$ from obtaining the new group key. This is the so-called stateless leave rekeying approach. However, the worst-case multicast overhead will be $|R^{(t)}|+m+1$ encrypted keys plus $(|R^{(t)}|+m+1)*2\log_2 n$ bits and would become prohibitively high when $|R^{(t+1)}|$ is big. Therefore, we would rather use a similar stateful leave rekeying approach as discussed in Section 5.1.2. Suppose that the ranks intervals that cover the subset U-L (instead of $S^{(t)}$-L as in above approach) are I_1, I_2,…, I_r. GC needs to generate a random group key $GK^{(t+1)}$, and for each rank interval I_j = $[j_1,\ j_m]$, create a cipher text $C_{I_j}' = \left\{GK^{(t+1)}\right\}_{b_{n-j_1+1}\oplus f_{j_m}\oplus GK^{(t)}}$. After creating all those cipher texts, GC sends the following rekey message by multicast:

GC⇒$S^{(t+1)}$: $I_1, I_2, \cdots, I_r, C_{I_1}', C_{I_2}', \cdots, C_{I_r}'$.

Every current member of $S^{(t)}$ except those in L can decrypt this message to obtain the new group key $GK^{(t+1)}$. Compared with the stateless leave rekeying algorithm, the worst-case multicast size is only m+1 encrypted keys plus $(m+1)*2\log_2 n$ bits.

For join rekeying, the computational cost of GC is $2n$+1 evaluations of a one-way hash function at most (for computing the exclusive key pairs for the joining members and the new group key) and that of each group member except the joining member is only one evaluation of a one-way hash function. The unicast size is only $3m$ keys in total and the multicast size is only one notification message. For leave rekeying, the computational cost of GC is $2n$ evaluations of a one-way hash function (for computing the exclusive key pairs for the departing members) plus m+1 encryptions at most. The computational cost of each group member is n evaluations of a one-way hash function at most plus 1 decryption.

5.3 Group Rekeying Based on the BHT Algorithm

5.3.1 A stateless group rekeying protocol — Protocol III

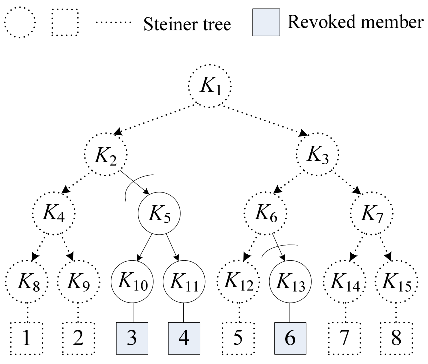

Fig. 3 Protocol III

Note that in the remainder of this paper, when referring to the FN 1st protocol, we mean the BHT-based version of FN 1st protocol. In Section 3.2, when using the CS protocol to construct a hybrid stateless protocol with collusion-bandwidth tradeoffs, we used the complete sub-tree method to help us find those subsets that cover legitimate users. Whereas in this section, we use the same method to help find those subsets that cover *illegitimate users* and their corresponding exclusive keys. Thus we improve the FN 1st protocol with respect to both computational overhead and security. For convenience, we call the improved protocol Protocol III. Suppose that there are changes in group membership (joins or leaves) at time t and the privileged set changes from $S^{(t)}$ to $S^{(t+1)}$. For convenience, we denote by ST(S) the Steiner tree that is spanning the root and all the u-nodes respectively associated with every member of a sub-set $S\subseteq U$. As illustrated in Figure 3, GC first computes a Steiner tree ST($S^{(t+1)}$) so as to obtain all those subsets that together cover all revoked users in $R^{(t+1)}$. According to the BHT algorithm, all exclusive keys for these subsets are just those associated with the roots of those sub-trees that hang off the Steiner tree ST($S^{(t+1)}$) as illustrated by Figure 3. Taking the case illustrated by Figure 3 as an example, these "covering" subsets are {3,4} and {6}, and their corresponding exclusive keys are K_5 and K_{13}, respectively. Suppose all these exclusive keys respectively corresponding to each of the covering subsets for $R^{(t+1)}$ are K_{i_1}, K_{i_2} ,…, and K_{i_m} . According to [2], the number of these keys is at most $|S^{(t+1)}|\log_2(n/|S^{(t+1)}|)$. GC then generates a random group key $GK^{(t+1)}$ and sends the following message by multicast:

GC⇒$S^{(t+1)}$: $i_1, i_2, \cdots, i_m, \left\{GK^{(t+1)}\right\}_{K_{i_1}\oplus K_{i_2}\oplus\cdots\oplus K_{i_m}}$.

Obviously, only those users in $S^{(t+1)}$ are able to decrypt this message and obtain the new group key according to the BHT algorithm. Compared with the CS protocol as illustrated by Figure 1, we compute the Steiner tree here to help find those "covering" subsets for $R^{(t+1)}$ rather than $S^{(t+1)}$.

Note that all exclusive keys used to encrypt the new group key are computed by GC from the root key K_1 on the fly when it performs group rekeying. As illustrated in Figure 3, the computational cost for GC to compute all exclusive keys is proportional to the size of the Steiner tree ST($R^{(t+1)}$) denoted by $|\mathrm{ST}(R^{(t+1)})|$. According to [10], we have $2|R^{(t+1)}|-1+\log_2(n/|R^{(t+1)}|)\le|\mathrm{ST}(R^{(t+1)})|\le 2|R^{(t+1)}|-1+|R^{(t+1)}|*\log_2(n/|R^{(t+1)}|)$. Intuitively, the minimum size of ST($R^{(t+1)}$) occurs when the ST($R^{(t+1)}$) is a densely embedded sub-tree of the key tree, as would happen when users in $R^{(t+1)}$ are associated with adjacent u-nodes of the key tree. The maximum size occurs when the ST($R^{(t+1)}$) is sparsely embedded in the key tree, as occurs when the u-nodes that users in $R^{(t+1)}$ are associated with are evenly distributed in the key tree. Since the size of the Steiner tree ST($R^{(t+1)}$) is determined by the distribution pattern of the revoked u-nodes in the key tree which is unpredictable due to group dynamics, we can only estimate the computational and communication cost of Protocol III (and the FN 1st protocol) by analyzing the following two extreme cases:

(1) The best case —

When ST($R^{(t+1)}$) has the minimum size, the computational cost of both GC and members reaches its lowest level. The minimum value of ST($R^{(t+1)}$) occurs when the ST($R^{(t+1)}$) includes a "densely packed" sub-tree of size $2|R^{(t+1)}|-1$ at the lowest level of the key tree, and this sub-tree is attached to the root of the key tree by a single path of length $\log_2(n/|R^{(t+1)}|)$. Therefore, the computational cost of GC amounts to $\log_2(n/|R^{(t+1)}|)$ evaluations of a one-way hash function plus 1 encryption. As introduced in Section 2.1, in the FN 1st protocol, the key used to encrypt the new group key is computed by

Xoring all leaf exclusive keys respectively associated with those revoked users in $R^{(t+1)}$. Therefore, for the FN 1st protocol, the computational cost of GC in this case amounts to $2|R^{(t+1)}|$-2+$\log_2(n/|R^{(t+1)}|)$ evaluations of a one-way hash function plus 1 encryption. For both Protocol III and the FN 1st protocol, because the computational cost of a member is determined by the relative position of its associated u-node with respect to the "densely packed" sub-tree, we only discuss the average cost here. For Protocol III, the average computational cost of a member is $\frac{n*\log_2\left(n/\left|R^{(t+1)}\right|\right)}{\left|S^{(t+1)}\right|}-2$ evaluations of a one-way hash function plus 1 decryption. For the FN 1st protocol, the average computational cost of a member is $\frac{n*\log_2\left(n/\left|R^{(t+1)}\right|\right)}{\left|S^{(t+1)}\right|}+2\left|R^{(t+1)}\right|-4$ evaluations of a one-way hash function plus 1 decryption. The multicast size is one encrypted key plus $\log_2(2n$-1) bits (note that it takes $\log_2(2n$-1) bits to uniquely identify a node of a full binary tree with n leaf nodes).

(2) The worst case —

When ST($R^{(t+1)}$) has the maximum size, the computational cost of both GC and members reaches its highest level. The maximum value of ST($R^{(t+1)}$) occurs when the ST($R^{(t+1)}$) includes a subtree of size $2|R^{(t+1)}|$-1 at the highest level of the key tree, with $|R^{(t+1)}|$ paths each of length $\log_2(n/|R^{(t+1)}|)$ flowing downward from its leaves to the leaves of the key tree. For both Protocol III′ and the FN 1st protocol, the computational cost of GC amounts to $2|R^{(t+1)}|$-2+$|R^{(t+1)}|*\log_2(n/|R^{(t+1)}|)$ evaluations of a one-way hash function plus 1 encryption and that of a member is $2|R^{(t+1)}|$-3+$|R^{(t+1)}|*\log_2(n/|R^{(t+1)}|)-2\log_2|R^{(t+1)}|$ evaluations of a one-way hash function at most plus 1 decryption. For both protocols, the multicast size is one encrypted key plus $|S^{(t+1)}|\log_2(n/|S^{(t+1)}|)*\log_2(2n$-1) bits at most (this is what it takes to identify all those exclusive keys used to encrypt the new group key, whose number is at most $|S^{(t+1)}|\log_2(n/|S^{(t+1)}|)$).

Although both Protocol III and the FN 1st protocol have the same worst-case computational and communication overhead, the advantage of Protocol III over the FN 1st protocol in both aspects increases as the revoked u-nodes become more densely distributed in the key tree, and reaches its maximum when all revoked u-nodes can be covered by just one subset.

Another interesting advantage of Protocol III over the FN 1st protocol is concerning collusion resistance. In the FN 1st protocol, the new group key is encrypted by a key computed by Xoring all the leaf exclusive keys respectively for each revoked user in $R^{(t+1)}$. An arbitrary pair of revoked users in $R^{(t+1)}$ can collectively compute all these exclusive keys from their personal keys, and thus can decrypt the rekey message. Unlike the FN 1st protocol, not an arbitrary pair of revoked users in

$R^{(t+1)}$ can collude to break the security of Protocol III. For example, a coalition among those revoked users in $R^{(t+1)}$ who belong to the same "covering" subset, e.g, S is useless because it is computationally infeasible for them to compute the corresponding exclusive key K_S that is used to encrypt the new group key $GK^{(t+1)}$ according to Theorem 3. Taking the case illustrated in Figure 3 as an example, a coalition between users 3 and 4 will be useless.

5.3.2 A stateful group rekeying protocol — Protocol III'

Here, we introduce the stateful group rekeying protocol that is based on the BHT algorithm — Protocol III'. Suppose that m users denoted by $J=\{i_1, i_2,\ldots, i_m\}$ join the group $S^{(t)}\subset U$ at the same time t. GD performs join rekeying similar to that of Protocol I' except that the unicast message is as follows:

For each $i_j\in J$ ($j=1,\ldots, m$)

GC→i_j: *sibling keys for* i_j, $GK^{(t+1)}$.

Both immediate and batch leave rekeying can be supported by using Protocol III straightly. Suppose that m users leave the group $S^{(t)}$ at the same time t and we denote this set of users by $L=\{i_1, i_2,\ldots, i_m\}$. GC simply computes the Steiner tree ST($R^{(t)}\cup L$) to obtain all subsets that together cover all revoked users in $R^{(t)}\cup L$, and use the corresponding exclusive keys for these subsets to encrypt the new group key so as to exclude all users in $R^{(t)}\cup L$ from obtaining the new group key. This is the so-called stateless leave rekeying approach. Here we also introduce a similar stateful leave rekeying approach as discussed in Section 5.1.2. GC needs to compute the Steiner tree ST(L) instead of ST($R^{(t)}\cup L$) as in above approach to obtain all subsets that together cover revoked users in L instead of $R^{(t)}\cup L$. Suppose all exclusive keys respectively associated with each of these subsets are K_{i_1} , K_{i_2} ,…, and K_{i_m} . GC then generates a random group key $GK^{(t+1)}$ and sends the following message by multicast:

GC⇒$S^{(t+1)}$: $i_1, i_2, \cdots, i_m, \left\{GK^{(t+1)}\right\}_{K_{i_1}\oplus K_{i_2}\oplus\cdots\oplus K_{i_m}\oplus GK^{(t)}}$.

In this way, all current members of $S^{(t)}$ except users in L can decrypt this message and obtain the new group key. Compared with the stateless approach, the stateful one reduces the computational complexity from $O(|\text{ST}(R^{(t)}\cup L)|)$ to $O(|\text{ST}(L)|)$ but weakens the collusion resistance at the same time. Suppose that a first group of users (resp. a second group of users) leave the group at time t_1 (resp. at a later time t_2), and these users (resp. the second group of users) can be covered by just one subset denoted by L_1 (resp. L_2) when GC performs batch leave group rekeying using the stateful approach at time t_1 (resp. at time t_2). Of course, we have $L_1\cap L_2=\varnothing$. We also suppose that these disjoint subsets L_1 and L_2 can be covered by just one subset denoted by L (i.e., $L_1\cup L_2=L$) when GC performs batch group rekeying using the stateless approach at time t_2. For stateful batch group rekeying, an arbitrary pair of users respectively from L_1 and L_2 can collude to break the group forward security as follows: since they collectively know the exclusive key K_{L2} for L_2 and the group key $GK^{(t2+1)}$, they can collude to decrypt the rekey message at time t_2 to obtain $GK^{(t2+1)}$. On the contrary, for stateless batch group rekeying, an arbitrary type of coalition among users in L will be useless because it is computationally infeasible for them to compute the encryption key K_L for $GK^{(t2+1)}$ according to Theorem 3.

For join rekeying, the computational cost of GC is $m*(2\log_2 n\text{-}1)+1$ evaluations of a one-way hash function at most (for computing the personal keys for the joining members and the new group key) and that of each group member except the joining member is only one evaluation of a one-way hash function. The unicast size is $m*(\log_2 n+1)$ keys in total and the multicast size is only one notification message. The computational and communication cost of the stateless leave rekeying algorithm is just the same to Protocol III. The computational and communication cost of the stateful leave rekeying algorithm can be easily derived from that of Protocol III. We also give its computational and communication cost for the two extreme cases like we did for Protocol III: (1) The best case — The computational cost of GC amounts to $\log_2(n/m)$ evaluations of a one-way hash function plus 1 encryption. The average computational cost of a member is $\frac{n*\log_2(n/m)}{n-m}-2$ evaluations of a one-way hash function plus 1 decryption. The multicast size is only one encrypted key plus $\log_2(2n\text{-}1)$ bits. (2) The worst case — The computational cost of GC amounts to $2m\text{-}2+m*\log_2(n/m)$ evaluations of a one-way hash function plus 1 encryption, and that of a member is $2m\text{-}3+m*\log_2(n/m)\text{-}2\log_2 m$ evaluations of a one-way hash function at most plus 1 decryption. The multicast size is one encrypted key plus $(n\text{-}m)*\log_2[n/(n\text{-}m)]*\log_2(2n\text{-}1)$ bits at most.

Remark 3: Compared with its corresponding stateless counterpart protocol, the stateful join rekeying algorithm (resp. stateful leave rekeying algorithm) requires every receiver to store the current group key $GK^{(t)}$ which will be used to derive (resp. encrypt) the next group key $GK^{(t+1)}$. If a legitimate receiver misses the current rekey message, she (or he) may be unable to decrypt all future rekey messages. That is why we call them "stateful".

6 Security Proofs

Panjwani [23] developed a symbolic security model for analyzing the security of multicast key distribution protocols. In this model, all keys and messages generated by a multicast key distribution protocol are treated as abstract data types and cryptographic primitives as abstract functions over such data types. Security can be specified by the notion of *recoverability*, i.e., some group key is safe if it cannot be recovered by an adversary from its personal key and all rekey messages. Below, we prove that all three protocols are secure against single-user attacks (1-resilient) under this model.

Consider a multicast group of n users, labelled by 1, 2,…, n. For an *n-user group rekeying protocols* Π, we introduce the following notations given by [23]. At any time t, the privileged set of users who are authorized to receive information sent over a multicast channel is denoted by $S^{(t)} \subseteq \{1,2,\ldots, n\}$. The rekey message generated by protocol Π (including both unicast and broadcast messages) for $S^{(t)}$ is denoted by $M^{\Pi}_{S^{(t)}}$. The group key used to encrypt all the information sent to $S^{(t)}$ is denoted by $GK^{(t)}$. Let $[n]$ denote the set $\{1,\ldots,n\}$ and let $2^{[n]}$ denote the power set of $[n]$. An arbitrary group dynamics up to time t can be uniquely represented by a sequence of privileged user sets $\vec{S}^{(t)} = (S^{(0)}, S^{(1)}, \cdots, S^{(t)}) \in (2^{[n]})^t$. A message sequence $\vec{S}^{(t)} \in (2^{[n]})^t$ is called *simple*, if for all $t \geq 1$, $S^{(t-1)}$ changes into $S^{(t)}$ through a single change in group membership. It is clear that arbitrary group membership updates can be simulated using simple sequence only. Let $M^{\Pi}_{\vec{S}^{(t)}}$ denote the set of all the rekey messages generated by protocol Π up to time t. That is, $M^{\Pi}_{\vec{S}^{(t)}} = \bigcup_{1 \leq t' \leq t} M^{\Pi}_{S(t')}$. Each user i obtains a personal key set PKS_i from the key server when it joins the group. For any information set M, we use $Rec(M)$ to denote the set of all information that are *recoverable* from M by using all sorts of cryptographic transformations employed by the group rekeying protocol (irrespective of the number of steps required to do so).

Definition 3: An n-user stateful group rekeying protocol Π is called *correct* if for all $t \geq 0$, and for all sequence $\vec{S}^{(t)} \in (2^{[n]})^t$ there exists a key $GK^{(t)}$ such that $\forall i \in S^{(t)} : GK^{(t)} \in \mathrm{Re}c(PKS_i \bigcup M^{\Pi}_{\vec{S}^{(t)}})$.

Definition 4: An n-user stateless group rekeying protocol Π is called *correct* if for all $t \geq 0$, and for all sequence $\vec{S}^{(t)} \in (2^{[n]})^t$ there exists a key $GK^{(t)}$ such that $\forall i \in S^{(t)} : GK^{(t)} \in \mathrm{Re}c(PKS_i \bigcup M^{\Pi}_{S^{(t)}})$.

Definition 5: An n-user immediate stateful group rekeying protocol Π is called *secure against single-user attacks* (i.e., *1-resilient*), if for all $t \geq 0$, and for all simple sequence $\vec{S}^{(t)} \in (2^{[n]})^t$, $\forall i \notin S^{(t)}$, $GK^{(t)} \notin \mathrm{Re}c(PKS_i \bigcup M^{\Pi}_{\vec{S}^{(t)}})$.

Definition 6: An n-user stateless group rekeying protocol Π is called *secure against single-user attacks* (i.e., *1-resilient*), if for all $t \geq 0$, and for all sequence $\vec{S}^{(t)} \in (2^{[n]})^t$, $\forall i \notin S^{(t)}$, $GK^{(t)} \notin \mathrm{Re}c(PKS_i \bigcup M^{\Pi}_{S^{(t)}})$.

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). For stateful protocols, we only prove security with respect to an arbitrary simple message sequence (i.e., security of immediate group rekeying versions of stateful protocols). Security of batch group rekeying versions can be proved using a similar argument. We first prove the correctness and security of Protocol III′.

Theorem 4: *Protocol III′ is correct and 1-resilient.*

Proof: We prove this claim 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 $t\text{-}1 \geq 0$, then it is true for t as well. According to Protocol III′, PKS_i is fixed and consisted of all the siblings keys in the key tree user i is entitled to. For any simple sequence $\vec{S}^{(t)} = (S^{(0)}, S^{(1)}, \cdots, S^{(t-1)}, S^{(t)})$, we only need to consider the recoverability of group keys from PKS_i and $M^{\Pi}_{\vec{S}^{(t)}}$ in the following cases:

Case 1 ($i\in S^{(t-1)} \wedge i\in S^{(t)}$, and $S^{(t-1)}$ changes into $S^{(t)}$ due to the other member's departure): From inductive hypothesis, i holds $GK^{(t-1)}$ as required by Definition 3. According to the BHT algorithm and the leave rekeying algorithm of Protocol III′, i can recover group key $GK^{(t)}$ from $\{GK^{(t-1)}, PKS_i, M^{III}_{S^{(t)}}\}$.

Case 2 ($i\in S^{(t-1)} \wedge i\in S^{(t)}$, and $S^{(t-1)}$ changes into $S^{(t)}$ due to some member's joining): From inductive hypothesis, i holds $GK^{(t-1)}$ as required by Definition 3. According to the join rekeying algorithm, i can compute $GK^{(t)}$ as $GK^{(t)}=h(GK^{(t-1)})$.

Case 3 ($i\notin S^{(t-1)} \wedge i\in S^{(t)}$): That is to say, i joins the group at time t. According to the join rekeying algorithm, every newly joining member i can recover $GK^{(t)}$ from the unicast message contained in $M^{III}_{S^{(t)}}$.

Case 4 ($i\in S^{(t-1)} \wedge i\notin S^{(t)}$): That is to say, i is revoked at time t. From the inductive hypothesis, i holds $GK^{(t-1)}$. According to the BHT algorithm and the leave rekeying algorithm, i can never recover $GK^{(t)}$ from $\{GK^{(t-1)}, PKS_i, M^{III}_{S^{(t)}}\}$, i.e., $GK^{(t)} \notin \mathrm{Re}c(PKS_i \cup M^{III}_{\vec{S}^{(t)}}) = \mathrm{Re}c(PKS_i \cup M^{III}_{\vec{S}^{(t-1)}})$.

Case 5 ($i\notin S^{(t-1)} \wedge i\notin S^{(t)}$): That is to say, i is evicted before time t-1. From the inductive hypothesis, i can never recover $GK^{(t-1)}$ from $M^{III}_{\vec{S}^{(t-1)}}$. Suppose that the leaf exclusive key for i is K_{i1}. However, $GK^{(t)}$ is encrypted by $GK^{(t-1)}\oplus K_{i1}$ in $M^{III}_{S^{(t)}}$, therefore i can never recover $GK^{(t)}$ from $M^{III}_{\vec{S}^{(t)}}$, i.e., $GK^{(t)} \notin \mathrm{Re}c(PKS_i \cup M^{III}_{\vec{S}^{(t)}})$ .Therefore, according to Definition 3 and Definition 5, Protocol III′ is correct and 1-resilient. □

Using a similar argument, we can prove Theorem 5 and Theorem 6.

Theorem 5: *Protocol I′ is correct and 1-resilient.*

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

For stateless protocols, we prove security with respect to an arbitrary message sequence. We first prove the correctness and security of Protocol II.

Theorem 7: *Protocol II is correct and 1-resilient.*

Proof: According to Definition 4 and Definition 6, the correctness and security of Protocol II immediately follows from the following claim —*for all $t\geq 0$, and for all sequence $\vec{S}^{(t)} \in (2^{[n]})^t$, for an arbitrary user i, i can recover $GK^{(t+1)}$ from $M^{II'}_{S^{(t)}}$ if and only if $i\in S^{(t)}$*. We first prove its sufficiency. Suppose that $M^{II'}_{S^{(t)}} = I_1, I_2, \cdots, I_r, C_{I_1}, C_{I_2}, \cdots, C_{I_r}$. If $i\in S^{(t)}$ (for simplicity, we assume its rank is also i) , then we have $\exists j$ ($1\leq j\leq r$) s.t. $i\in I_j$, i.e., $j_1\leq i\leq j_m$. According to the DHC algorithm, we have $PKS_i=\{f_i, b_{n-i+1}\}$.

Because of $j_1 \leq i \leq j_m$, user i can derive both b_{n-j_1+1} from b_{n-i+1} and f_{j_m} from f_i. Therefore, i can recover $GK^{(t+1)}$ from $C_{I_j} = \left\{GK^{(t+1)}\right\}_{b_{n-j_1+1} \oplus f_{j_m}}$. Next, we prove its necessity. If user i can recover $GK^{(t+1)}$ from $M^{II'}_{S^{(t)}}$, then $\exists j$ $(1 \leq j \leq r)$ s.t. i can recover $GK^{(t+1)}$ from C_{I_j}. Therefore, i must know both b_{n-j_1+1} and f_{j_m}. However, according to the DHC algorithm, b_{n-j_1+1} is derivable by any user with rank no less than j_1, and f_{j_m} is derivable by any user with rank no more than j_m. That is to say, only user with rank between j_1 and j_m can derive both b_{n-j_1+1} and f_{j_m}. Therefore, it must be true that $i \in I_j$ $(j_1 \leq i \leq j_m)$, i.e., $i \in S^{(t)}$. □

Using a similar argument, we can prove Theorem 8 and Theorem 9.

Theorem 8: *Protocol I is correct and 1-resilient.*

Theorem 9: *Protocol III is correct and 1-resilient.*

7 Performance Comparisons of Related 1-Resilient Group Rekeying Protocols

7.1 A Comparison of Stateful 1-Resilient Group Rekeying Protocols

TABLE 4

COMPARISON OF RELATED IMMEDIATE GROUP REKEYING PROTOCOLS

Measures \ Protocols		Flat-table Protocol	LORE	Protocol I'	Protocol II'	Protocol III'
GC stora.		$2log_2 n$ keys	2 keys	n keys	2 keys	1 key
Member stora.		$log_2 n$ keys	2 keys	n-1 keys	2 keys	$log_2 n$ keys
Join Rekeying	GC comp.	$log_2 n * C_h$	$1C_E + n * C_h$	$1C_h$	$(n+1) * C_h$	$2\log_2 n * C_h$
	member comp.	$(log_2 n-1) * C_h$ at most	$1C_D$	$1C_h$	$1C_h$	$1C_h$
	Unicast size	$log_2 n+1$ keys	3 keys	n keys	3 keys	$log_2 n+1$ keys
	Multicast size	1 notifi. msg.	1 key	1 notifi. msg.	1 notifi. msg.	1 notifi. msg.
Stateful Leave Rekeying	GC comp.	$3log_2 n * C_E$	$4C_E + n * C_h$	$1C_E$	$2C_E + n * C_h$	$1C_E + log_2 n * C_h$
	member comp.	$(2log_2 n-1) * C_D$ at most	$2C_D + n * C_h$ at most	$1C_D$	$1C_D + n * C_h$ at most	$1C_D + [n * log_2 n/(n-1)-2] * C_h$ in average
	Multicast size	$log_2 n$ bits+$2log_2 n$ keys	$log_2 n$ bits+2 keys	$log_2 n$ bits+1 key	$log_2 n$ bits+2 keys	$log_2(2n-1)$ bits+1 key

C_E, C_D, C_h denote the computation cost of 1 block encryption, 1 block decryption, and 1 hashing, respectively. We use n to denote the prospective size of a group and it remains unchanged.

All the stateful 1-resilient group rekeying protocols introduced in Section 2 are falling into the category of immediate group rekeying. Therefore, we have to compare the immediate group rekeying versions of our protocols with them. The personal key assignment of the KHYC protocol [21] relies on a dynamic group access control structure [25] which keeps changing with group dynamics, while the LORE protocol [17], the flat-table protocol [12],[22], and the three proposed stateful protocols here all rely on static group access control structures which remain unchanged irrespective of group dynamics. It becomes very complicated to make a performance comparison between protocols with dynamic group access control structure and those with static one. We refer the interested readers to [25] for more information. Due to space constraint, we only make a performance comparison among all 1-resilient stateful group rekeying protocols with static group access control structures. We summarize those performance-related discussions already made in Sections 2, 4 and 5 to provide in a Table 4 a comprehensive comparison among these protocols covering the following measures: GC storage, member storage, GC computational overhead, member's computational overhead, unicast size, and multicast size.

Consider join rekeying. The Flat-table protocol has a worst-case computational overhead logarithmic in n for group members, whereas the others only have a constant overhead because the former requires each current member to update a part of its personal keys when a new member joins. Both the LORE protocol and Protocol II′ relies on a linear structure (a dual hash chain), and it takes them a computational overhead linear in n for GC to compute the personal key for the joining member.

Consider leave rekeying. The Flat-table protocol has the biggest multicast size and highest computational overhead for GC of all five protocols, since it requires GC to update all $\log_2 n$ auxiliary keys held by the leaving member and secretly distribute them to the remaining members. While the other four protocols all achieve a constant multicast size (the identification information for a revoked user is negligible, and thus its cost is usually not counted in the literature). Since both the LORE protocol and Protocol II′ rely on a linear structure with functional dependency (a dual hash chain), both have a computational complexity linear to group size n for both GC and members. Whereas, both the flat-table protocol and Protocol III′ rely on a tree structure, and thus have a computational complexity logarithmic in group size n for both GC and members.

Protocol III′ achieves both a logarithmic computational complexity and a logarithmic storage complexity while maintaining a constant multicast size. It achieves the most balanced performance, and is the best 1-resilient immediate group rekeying protocol of all the five protocols.

TABLE 5

COMPARISON OF BATCH GROUP REKEYING PROTOCOLS

Protocols Measures		Protocol I'	Protocol II'	Protocol III'	
GC stora.		n keys	2 keys	1 key	
Member stora.		n-1 keys	2 keys	$log_2 n$ keys	
Join Rekeying	GC comp.	$1C_h$	$2n*C_h$ at most	$[m-2+2m*\log_2(n/m)]*C_h$ at most	
	member comp.	$1C_h$	$1C_h$	$1C_h$	
	Unicast size	$m*n$ keys	$3m$ keys	$m*(log_2 n+1)$ keys	
	Multicast size	1 notifi. msg.	1 notifi. msg.	1 notifi. msg.	
Stateful Leave Rekeying	GC comp.	$1C_E$	$(m+1)*C_E + 2n*C_h$ at most	Best case	$1C_E + log_2(n/m)*C_h$
				Worst case	$1C_E+[2m-2+m*\log_2(n/m)]*C_h$
	member comp.	$1C_D$	$1C_D+n*C_h$ at most	Best case	$1C_D+[n*log_2(n/m)/(n-m)-2]*C_h$ in average
				Worst case	$1C_D+[2m-3+m*\log_2(n/m)-2\log_2 m]*C_h$ at most
	Multicast size	$m*log_2 n$ bits+1 key	$2(m+1)log_2 n$ bits+$(m+1)$ keys	Best case	$log_2(2n-1)$ bits+1 key
				Worst case	$(n-m)*log_2[n/(n-m)]*log_2(2n-1)$ bits+1 key at most

Denotations of C_E, C_D, C_h, n are same as Table 4. We use m to denote the number of users who join/leave the group at the same time.

We also provide in Table 5 a comparison of all three batch group rekeying protocols proposed in Section 5. For the same reason discussed above, Protocol III′ achieves the most balanced performance.

7.2 A Comparison of Stateless 1-Resilient Group Rekeying Protocols

TABLE 6

COMPARISON OF RELATED 1-RESILIENT STATELESS PROTOCOLS

Protocols / Measures	**FN 2nd Protocol**	**Protocol I**	**Protocol II**	**FN 1st Protocol**		**Protocol III**	
Collusion between any two users in $R^{(t+1)}$	Yes	Yes	Yes	Yes		No	
GC stora.	1 private key, n public keys	n keys	2 keys	1 key		1 key	
Member stora.	1 private key, n public keys	n- 1 *keys*	2 keys	$log_2 n$ keys		$log_2 n$ keys	
GC comp.	$\vert S^{(t+1)}\vert * C_{Ex}$	$1C_E$	$(\vert R^{(t+1)}\vert + 1)*C_E +$ $2n * C_h$ at most	Best case	$1C_E+[2\vert R^{(t+1)}\vert-2+log_2(n/\vert R^{(t+1)}\vert)]*C_h$	Best case	$1C_E+log_2(n/\vert R^{(t+1)}\vert)*C_h$
				Worst case	$1C_E+[2\vert R^{(t+1)}\vert-2+\vert R^{(t+1)}\vert*log_2(n/\vert R^{(t+1)}\vert)]*C_h$	Worst case	$1C_E+[2\vert R^{(t+1)}\vert-2+\vert R^{(t+1)}\vert*log_2(n/\vert R^{(t+1)}\vert)]*C_h$
Member comp.	$\vert S^{(t+1)}\vert * C_{Ex}$	$1C_D$	$1*C_D+n*C_h$ at most	Best case	$1C_D+[n*log_2(n/\vert R^{(t+1)}\vert)/\vert S^{(t+1)}\vert+2\vert R^{(t+1)}\vert-4]*C_h$ in average	Best case	$1C_D+[n*log_2(n/\vert R^{(t+1)}\vert)/\vert S^{(t+1)}\vert-2]*C_h$ in average
				Worst case	$1C_D+[2\vert R^{(t+1)}\vert-3+\vert R^{(t+1)}\vert*log_2(n/\vert R^{(t+1)}\vert)-2\log_2\vert R^{(t+1)}\vert]*C_h$ at most	Worst case	$1C_D+[2\vert R^{(t+1)}\vert-3+\vert R^{(t+1)}\vert*log_2(n/\vert R^{(t+1)}\vert)-2\log_2\vert R^{(t+1)}\vert]*C_h$ at most
Multicast size	$\vert R^{(t+1)}\vert*log_2 n$ bits	$\vert R^{(t+1)}\vert*log_2 n$ bits+ 1 key	$2(\vert R^{(t)}\vert+1)*log_2 n\ bits+$ $(\vert R^{(t+1)}\vert+1)*\ keys$ at most	Best case	$\vert R^{(t+1)}\vert*log_2(2n-1)$ bits+1key	Best case	$log_2(2n-1)$ bits+1key
				Worst case	$\vert S^{(t+1)}\vert log_2(n/\vert S^{(t+1)}\vert)*log_2(2n-1)$ bits+1 key at most	Worst case	$\vert S^{(t+1)}\vert log_2(n/\vert S^{(t+1)}\vert)*log_2(2n-1)$ bits+1 key at most

Denotations of C_E, C_D, C_h, n are same as Table 4. We also use C_{Ex} denote the computation cost of a modular exponentiation operation. Recall that we use $S^{(t)}$ to denote the set of legitimate users at time t and $|S^{(t)}|$ to denote the size of $S^{(t)}$. $R^{(t)}$ denotes the complement of set $S^{(t)}$.

In this section, we summarize those performance-related discussions already made in Sections 2, 4, and 5 to provide in Table 6 a comprehensive comparison between our stateless protocols and other existing 1-resilient stateless group rekeying protocols, i.e., the FN 1st protocol [1] and the FN 2nd protocol [1]. The FN 2nd protocol requires both GC and member to perform $|S^{(t)}|$ modular exponentiations ($|R^{(t)}| << |S^{(t)}|$ in most occasions) and thus has the highest computational overhead of all five protocols. Although Protocol I′ has the constant computational overhead, it has a storage overhead linear to the group size. The FN 1st protocol and Protocol III achieve both a logarithmic computational complexity and a logarithmic storage complexity while maintaining a constant multicast size if we don't count the set identification information. Although both Protocol III and the FN 1st protocol have the same worst-case computational and communication overhead, the former is superior to the latter in the best case. As we discussed in Section 5.3.1, although both Protocol III and the FN 1st protocol have the same worst-case computational and communication overhead, the advantage of Protocol III over the FN 1st protocol in both aspects increases as the revoked u-nodes become more densely distributed in the key tree, and achieves its maximum in the best case. As discussed in Section 5.3.1, another advantage of Protocol III over all the other protocols lies in its collusion resistance. To the best of our knowledge, Protocol III is the most efficient and secure 1-resilient stateless group rekeying protocol to date. According to both Table 5 and Table 6, each stateless leave rekeying algorithm relying on the corresponding stateless protocol usually has a higher computational overhead and a bigger multicast size than its stateful counterpart.

8 Conclusion and Future Research

We clarified the meaning of research on 1-resilient group rekeying protocols by showing that they are actually building blocks for constructing hybrid group rekeying protocols with tunable collusion-bandwidth tradeoffs. We proposed three personal key assignment algorithms based on the idea of exclusive key. Employing each of them, we proposed both a stateful group rekeying protocol and a stateless one. Each of all three stateful protocols (resp. all three stateless protocols) offers a different tradeoff between storage overhead and computational overhead. Micciancio and Panjwani [16] successfully developed a flexible symbolic model of computation and used it to prove lower bounds on the communication complexity of generic collusion-resistant multicast key distribution protocols. For 1-resilient group rekeying protocols with constant communication overhead, it would be interesting to further apply their method to derive a lower bound on either of computational complexity and storage complexity when the other is constrained. Fiat and Naor [1] utilized 1-resilient BE protocols to construct k-resilient BE protocols. It will be a worthwhile topic to construct from 1-resilient stateful group rekeying protocols k-resilient ones with a communication complexity lower than $O(\log_2 n)$.

References

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

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

[3] 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.

[4] 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.

[5] 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, 2010.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[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 Journal on Selected Areas in Communications,* vol. 17, no. 9, pp. 1614-1631, 1999.

[13] S. Setia, S. Koussih, S. Jajodia, and E. Harder, “Kronos: A scalable group re-keying approach for secure multicast,” *Proc. IEEE Symposium on Security and Privacy*, pp. 215-228, 2000.

[14] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam, “Batch rekeying for secure group communications,” *Proc. 10th international conference on World Wide Web*, pp. 525-534, 2001.

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

[16] D. Micciancio, and S. Panjwani, “Optimal communication complexity of generic multicast key distribution,” *IEEE-ACM Trans. Networking,* vol. 16, no. 4, pp. 803-813, 2008.

[17] 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.

[18] 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.

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

[20] NIST, “Secure Hash Standard,” *FIPS Publication 180-1*, Apr. 1995.

[21] 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 (ITCC),* Vol 1, 2005, pp. 685-690.

[22] 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*, vol.2, pp. 689-698 , 1999

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

[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, Q. Huang, B. Yang, and Y. Zhang, “Efficient Multicast Key Distribution Using HOWP-based Dynamic Group Access Structures,” *Cryptology ePrint Archive, Report 2011/576*, 2011.

Jing Liu received the Ph.D. degree in computer application technology from University of Electronic Science and Technology of China, Chengdu, China, in 2003. From September 2003 to July 2005, he was with No.30 Institute of China Electronics Technology Group Corporation, Chengdu, China, as a postdoctoral fellow. Since 2005, he has been a lecturer at School of Information Science and Technology, Sun Yat-Sen University, China. He has also been affiliated with Guangdong Key Laboratory of Information Security and Technology, Guangzhou, China, since 2005. His current research interests mainly focus on applied cryptography and network security.

Changji Wang is an associate professor at the School of Information Security and Technology at the Sun Yat-sen University, Guangzhou, China. He completed his PhD degree from Chinese Academy of Sciences, Beijing in 2002. From 2002 to 2004, he was a postdoctoral research fellow in Network Research Center of Tsinghua University. From 2009 to 2010, he was a visiting scholar in Information Security Institute, Queensland University of Technology, Australia. Since 1999 he has been researching in cryptography and network security.