Performance Comparison of Stateful and Stateless Group Rekeying Algorithms

W. Chen & L. R. Dondeti

CMPSCI TR 03-32

Performance Comparison of Stateful and Stateless Group Rekeying Algorithms [*]

Weifeng Chen [†] and Lakshminath R. Dondeti [‡]

Technical Report 2003-32
Department of Computer Science
University of Massachusetts, Amherst

Abstract

Scalable group rekeying schemes proposed in the literature can be classified into three categories: stateful schemes, stateless schemes and self-healing schemes. They differ mainly on the interdependency of rekey messages and messaging overhead in rekeying. Logical key hierarchy (LKH) based approaches are stateful in that members should have received past rekeying messages to decrypt current rekeying messages. Stateless rekeying algorithms, such as subset difference based member revocation (SDR) mechanism, on the other hand, use keys sent during member registration/initialization to encrypt the group key. In other words rekeying messages are independent of each other and consequently members going offline can decrypt the group key without having to consult the group manager. This is an important property considering that reliable delivery of rekey messages is a significant issue in deploying group and multicast security solutions. While in self-healing schemes, a rekeying message contains not only the current key, but also the shares of previous and future keys such that a member can recover a missed key by combining corresponding shares received by the member through other rekeying messages.

SDR messaging overhead in rekeying is dependent on the membership during an entire multicast session whereas LKH messaging overhead is dependent on membership of the group during a rekeying instance. In this paper, we study the advantages and applicability of stateful and stateless rekeying algorithms to different groups and multicast security applications. We analytically compare the storage cost and the rekeying cost (number of unit-size encrypted messages) of LKH and SDR in immediate and batch rekeying scenarios. We implemented the two algorithms and simulated different membership scenarios to compare the rekeying cost. The simulation study shows that LKH performs better in immediate rekeying and small batch rekeying, whereas stateless rekeying performs better as we process membership changes in larger batches. In some cases, stateless rekeying was observed to be as inefficient as encrypting the group key separately for each member of the group. We also report on the effect of member adjacency on SDR rekeying cost that it seems to have more impact on rekeying cost than the number of membership changes.

[*]This work is supported by the Defense Advanced Research Projects Agency (DARPA) under contract N66001-00-C- 8011.

[†]Department of Computer Science, University of Massachusetts, Amherst, MA 01003. *chenwf@cs.umass.edu*

[‡]Strategic protocols group, Nortel Networks, 600 Technology Park Drive, MS E65-60-202, Billerica, MA 01821. *ldondeti@nortelnetworks.com*

1 Introduction

The dramatic growth of profitable services on the Internet, such as Pay-Per-View, stock quote distribution, has been benefited from the successful deployment of secure communication. The IETF multicast security working group identified three problem areas in secure group communication, viz., key distribution, data origin authentication, and policy management. Group key distribution facilitates confidential group communication as well as enforcement of access control. A *group key* k_g, thus, is introduced to encrypt group data. Strict group privacy requires that a host be able to decrypt group communications only when it is a member of the group. Thus, when a new member joins, k_g needs to be updated in order to prevent the new member from accessing past communications in the group. This property is known as *backward access control*. The group key also needs to be changed to prevent departing members from accessing future data and enforce *forward access control*. Although it is easy to enforce backward access control – by multicasting the new group key, $k_g^{'}$, encrypted using the current k_g – it is difficult to ensure forward access control. Typically, we need to encrypt $k_g^{'}$ with a common key (or set of keys) known only to the remaining membership. Many algorithms [2, 7, 8, 10, 15] have been proposed for scalable rekeying of large groups (in the order of thousands or more of members). Several surveys of group rekeying algorithms are available in the literature [9, 11]. In the rest of this paper, we will refer to an entity called *Group Controller and Key Server (GCKS)*, by the IETF MSEC framework for dealing with key distribution and rekeying.

1.1 Classification of group rekeying algorithms

Based on the interdependency of rekeying messages, group key management algorithms can be classified into stateless ([10]), stateful ([2, 7, 15]) and self-healing rekeying schemes ([13]).

In stateless rekeying, rekey messages are encrypted only by the keys distributed during member registration. This allows rekey messages to be independent of each other. But in stateful schemes, the GCKS uses key encryption keys (KEKs) sent in a given rekeying instance to encrypt the keys to be sent in the next (or future) rekeying instance(s). Finally in self-healing schemes, the GCKS sends out not only the updated group key, but also some redundant information of previous group keys or future group keys, in a rekeying message. Active members who miss a particular group key are able to derive it using the redundant information embedded in other rekeying messages.

In stateful rekeying, the GCKS uses keys sent in a given rekeying instance to encrypt the keys to be sent in the next or future rekeying instance(s). Thus, a member that goes offline during a rekeying instance may not be able to decrypt any future rekey messages. In other words, authorized members may be inadvertently excluded from the group when stateful rekeying is employed. Consequently, reliable transport of rekey messages is a requirement for effective use of stateful rekeying. Furthermore, some members may have to contact the GCKS for re-synchronization with the rekeying mechanism. Rekeying algorithms based on a logical hierarchy of KEKs fall into this category.

When stateless rekeying is employed, a member that goes offline misses only the data and the keys sent while it was offline. Once online, the member can decrypt future keys and data without having to contact the GCKS. Note however that stateless rekeying is not a substitute for reliable delivery of rekey messages. More specifi cally, if secure multicast data is being sent over a reliable channel, group keys must also be sent using a reliable delivery mechanism. Otherwise, it is plausible that members may receive data, but miss the group key(s) required to decipher the data. The advantage of stateless rekeying is that the GCKS has some leeway if the application tolerates loss of data in sending rekey messages. In contrast, if stateful rekeying (e.g. LKH) is used, reliable transmission of rekey messages is necessary even if the multicast application tolerates data losses. In [10], D. Naor, *et al.* describe two stateless rekeying schemes: Complete Subtree Revocation (CSR) and Subset Difference Revocation (SDR). SDR is the more effi cient approach among the two.

Stateless rekeying comes at a cost however. More specifi cally, *rekeying cost*, measured by the number of unit-size encrypted messages during rekeying, in SDR could be more than that in LKH. Analytical cost comparison of these two schemes has been done elsewhere [10], but an important difference between the two schemes was overlooked in that study. LKH rekeying is based on instantaneous membership in the group, whereas SDR rekeying is based on total membership of the group during the entire secure multicast session, and the revoked member set at the time of rekeying. These differences are highlighted in this paper.

Active members who miss a stateful group rekey message may request the GCKS for retransmission by sending a NACK [16], or a request for re-synchronization. This is a possibility in stateless rekeying as well since although members can correctly decrypt rekeying messages even after being offline for a while, they cannot recover the missed group keys without contacting the GCKS. Thus members need a back channel to GCKS in both stateful and stateless scheme to achieve *reliable* rekeying.

However, this is not the case in the self-healing schemes, whereby members can derive the lost keys without contacting the GCKS. Particularly, the GCKS predesigns a series of group keys, each for a rekeying instance. Each rekeying message not only contains the current group key, but also embeds the redundant information, referred to as *share* in [13], for previous and future group keys. These shares can then be used to recover the missed group keys. This property makes self-healing attractive in applications where back channels do not exist, or expensive (e.g. satellite broadcast TV). But rekeying using the self-healing scheme is computationally expensive and may be susceptible to collusion attacks from excluded members.

In this paper, we focus on the messaging overhead of the rekeying schemes that do not include retransmissions to cover packet losses. Thus we assume that there is no packet loss when rekeying. Specifi cally, we only compare LKH and SDR. We exclude the self-healing scheme from this study for it is clearly ineffi cient compared to LKH and SDR and is susceptible to collusion attacks from non-members.

Considering that rekeying cost in LKH and SDR is dependent on members' positions in the key trees, and on the key tree sizes, we use simulation for the comparison study. We use both real-life group membership data and DaSSF [5] generated data. Our simulation result shows that LKH requires fewer rekeying messages than SDR in immediate rekeying and in small batch rekeying. However, when the batch size increases, SDR outperforms LKH. And we also observe that SDR performs better in large groups than small groups. This study helps us understand the respective advantages of these two schemes, and provides guidelines for applications in choosing the appropriate rekeying scheme according to the group membership behavior and rekeying policy.

The rest of this paper is organized as follows. In Section 2, we briefly overview LKH and SDR. We present an analytical comparison between LKH and SDR in Section 3. Section 4 describes our implementation and simulation. Section 5 concludes the paper and outlines our future work.

2 LKH and SDR

In LKH [14, 15], the GCKS organizes the keys in a logical hierarchy. The root node of the key tree represents the group key and each of other nodes corresponds a KEK. Each active member in the group is assigned a leaf node and receives all the KEKs of nodes along the path from its leaf to the root. Consequently, the group key is known to all active members and the leaf nodes represent the unique keys that each member shares with the GCKS. When a member joins or leaves the group, in order to keep the backward/forward access control, all the keys in its path to the root must be changed, each of which is encrypted by the KEKs corresponding to the node's children and sent out. Since KEKs themselves may be updated and future rekey messages may be encrypted using the changed KEKs sent in the current rekey message, a member may not be able to decrypt future rekey messages to retrieve the group data keys if he/she does not receive the current rekeying message. But this scheme is effi cient since only as many keys as the height of the tree change during rekeying. For efficient operation, the size of the LKH key tree is maintained to be just enough to hold all active members currently in the group. Generally, the key tree is *contracted* (or *expanded*) when members leave (or join) the group. Furthermore, keeping the LKH key tree balanced so as to achieve the minimum height of the key tree can reduce the number of keys to be updated during rekeying.

Similar to LKH, SDR constructs a binary key tree and each node in the binary tree corresponds a key. Each member is also assigned a leaf node, whose position determines the set of keys distributed to the member. Active members during an instance of rekeying are divided into the predefined subsets (as we will detail shortly), and the group key is sent encrypted with the keys of resultant subsets. The keys of the resultant subsets, the KEKs in SDR, can be computed from the key sets that members receive during registration. SDR is a stateless scheme because the key sets, thus the KEKs, are not changed during rekeying. As a consequence, once a leaf node in the SDR key tree has been assigned to a member, it cannot be assigned to any other members even when that member leaves the group. This results in the SDR key tree requirement of being large enough to hold all unique members; thus no dynamical adjustment (contracting or expanding) of the key tree can be done.

In the rest of this paper, a leaf node in the key tree and the member assigned to that node are treated indistinguishable when there is no risk of ambiguity.

2.1 Key tree size

The key tree size is an important difference, which is overlooked in a quick comparison of LKH and SDR in [10].

Due to the fact that all the keys known to (or to be known to) departing (or joining) members are changed during rekeying, LKH allows reassignment of an empty tree node position left by a departing member, to a joining member. Additionally, recalling that rekeying overhead is dependent on the height of the key tree, it is efficient to dynamically contract or expand the key tree. The LKH key tree is thus maintained to just hold current members in the group. Some algorithms have been proposed to dynamically keep the tree full and balanced [4, 16].

However, each leaf node in the SDR key tree cannot be assigned to more than one member, for the corresponding keys are unchanged. The SDR key tree is thus larger than the LKH key tree. In particular, the SDR key tree should be large enough to hold all potential members. Typically, the size of all potential members is estimated and then the SDR key tree is constructed in advance. The number of potential members is dependent on the length of the *multicast session*, e.g., a military battle or a baseball game on Pay-Per-View. Generally speaking, the longer the session is, the larger the size of potential members is. Thus, when we compare the performance of these two schemes, we consider a particular multicast session assuming that the size of potential members of the session is estimated correctly. When the multicast session is given, the difference between the size of a LKH key tree and a SDR key tree obviously depends on the relationship between the number of potential members and concurrent members. This relationship is application-specifi c, as we will see in this paper.

2.2 Immediate and batch rekeying

Given a multicast session, the total number of rekeying instances in the whole session varies depending on the rekeying strategies. A GCKS may process membership changes immediately or in a batch. *Immediate rekeying* results in excessive computation and communication overhead but provides strict access control. It is preferred in the applications where strict confi dentiality is required (e.g., military applications). *Batch rekeying* is then proposed to reduce the excessive overhead but with loss of strict access control [12]. Two types of thresholds can be used for batch rekeying. In the fi rst, the GCKS rekeys after a fi xed time period (T) and in the second, it rekeys after a fi xed number of members have left the group. We refer to these variations as *periodic batch rekeying* and *membership batch rekeying*, respectively. The lower overhead makes batch rekeying attractive for commercial applications that use secure multicasting to facilitate periodic billing (e.g., 30-minute segments of TV programming). We will consider both periodic and membership batch rekeying strategies in our comparison between LKH and SDR.

Recently, an *exposure-oriented rekeying* scheme is proposed to consider the cost of *not* rekeying [17]. This paper focuses on the rekeying cost and does not include the exposure-oriented rekeying.

2.3 Symbols used in the paper

Some notation need to be introduced for our further discussion.

- N: the set of potential members in the given multicast session. This is only used in SDR. N=$|\mathbf{N}|$.
- m: the set of members to stay in the group after the rekeying, referred to as *remaining members*. m=$|\mathbf{m}|$.
- j: the set of joining members since last rekeying. It may be empty in the case where the GCKS

conducts an immediate rekeying when a member leaves the group. j=|$\mathbf{j}$|.

- $\mathbf{r}$: the set of departing members since last rekeying. Similarly, $\mathbf{r}$ may be empty. r=|$\mathbf{r}$|.

- $\mathbf{n}$: the set of members at the time when the GCKS is going to perform rekeying, i.e., $\mathbf{n} = \mathbf{m} + \mathbf{r}$. n=|$\mathbf{n}$|.

- $\mathbf{R}$: all potential members except those to be in the group after the rekeying, i.e. $\mathbf{R} = \mathbf{N} - \mathbf{m} = \mathbf{N} - \mathbf{n} + \mathbf{r}$. $\mathbf{R}$ is only used in SDR. R=|$\mathbf{R}$|.

- $\{k_g\}_{k_{sg}}$: k_g is encrypted with k_{sg}.

Figures 1 and 2 show the snapshots of a LKH and an SDR key tree in two consecutive rekeying instances [1], where $\mathbf{N} = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}$. Specifi cally, in Figure 1(b) and 2(b), $\mathbf{n} = \{m_2, m_5, m_7, m_8\}$, $\mathbf{m} = \{m_2, m_5, m_8\}$, $\mathbf{j} = \emptyset$, $\mathbf{r} = \{m_7\}$ and $\mathbf{R} = \{m_1, m_3, m_4, m_6, m_7\}$. As we will see, the performance of SDR is dependent on N and R whereas that of LKH is dependent on n, j and r.

To clearly differentiate the overhead of LKH and SDR, we illustrate the rekeying procedures of these two schemes in the following two subsections.

2.4 Illustration of LKH-based rekeying

Using LKH, the keys to be updated during rekeying are encrypted by KEKs and multicast. There are three approaches to organize the rekeying messages: *user-oriented, key-oriented* and *group-oriented* [15], which result in different number of rekeying messages. For example, in Figure 1(a), when members m_1 and m_4 depart, $k_{1,2}$, $k_{4,5}$, $k_{1,5}$ and k_g need to be updated. With key-oriented rekeying, GCKS sends out the following rekeying messages: $\{k_{2,5}\}_{k_2}$, $\{k_{2,5}\}_{k_5}$, $\{k_g'\}_{k_{2,5}}$, and $\{k_g'\}_{k_{7,8}}$; 4 messages in all. Note that the LKH key tree in Figure 1(b) has been contracted. Using user-oriented and group-oriented approaches, the number of rekeying messages for this particular rekeying is 3 and 1 respectively. It should be noted that the size of each message varies in these three approaches. Thus it is not appropriate to measure the rekeying cost just using the number of rekeying messages. Instead, the size of each message needs to be considered. We use the number of unit-size encrypted messages to measure the rekeying cost. Specifi cally, if a message contains k encrypted keys, the message incurs k units cost. The cost associated with a rekeying instance is then the total cost of all the rekeying messages required to fulfi ll the rekeying. In this sense, the number of rekeying

[1]The degree of the LKH key tree may be varied. We only demonstrate the binary one although we also consider other degrees in Section 4.

messages using key-oriented approach in LKH is exact the number of unit-size encrypted messages. Also note that key-oriented approach has the minimum unit-size cost among the three approaches [6].

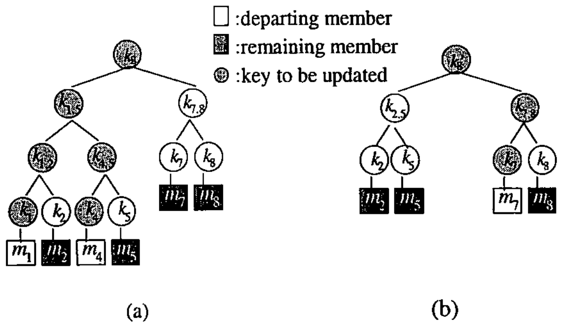

Figure 1: Stateful rekeying of LKH.

Figure 1 also illustrates the stateful property of LKH rekeying. In order to be able to decipher the rekeying messages when member m_7 leaves (Figure 1(b)), members m_2 and m_5 need to have $k_{2,5}$, which was sent when members m_1 and m_4 left (shown in Figure 1(a)). Thus failure to receive past rekeying messages may affect members' ability to decipher current rekeying messages. It should be pointed out that some members missing some rekeying messages may still be able to decrypt future messages. This occurs when the missed messages did not contain updated KEKs known to the members. For example, in Figure 1(a), if m_7 and m_8 miss the rekeying messages when members m_1 and m_4 depart, they are still able to decipher future rekeying messages since the KEKs known to m_7 and m_8 (k_7, k_8 and $k_{7,8}$) are not updated.

2.5 Illustration of rekeying using SDR

Although SDR constructs a binary key tree similarly with LKH, the SDR rekeying is quite different.

As we mentioned before, in SDR, a tree large enough to hold N potential members is constructed in advance and members are assigned to leaf nodes. For simplicity, we assume that $\log_2 N$ is an integer that the SDR key tree is balanced. When the GCKS is going to rekey at time t, there are three types of leaf nodes in an SDR tree: nodes that correspond to remaining members $\mathbf{m}_t$; nodes that correspond departing members since last rekeying, denoted as $\mathbf{r}_t$; and nodes not yet assigned, called $\mathbf{e}_t$. Directly, it follows $\mathbf{n}_t = \mathbf{m}_t + \mathbf{r}_t$ and $\mathbf{R}_t = \mathbf{N} - \mathbf{m}_t$. For instance, in Figure 2(a), $\mathbf{m}_t$ = {8, 11, 13, 14}, $\mathbf{r}_t$ ={7, 10}, $\mathbf{e}_t$ = {9, 12} and $\mathbf{R}_t = \{7, 9, 10, 12\}$. In the rest of this paper, $\mathbf{R}_t$ and $\mathbf{n}_t$ are simplifi ed as $\mathbf{R}$ and $\mathbf{n}$.

During each rekeying, it is required that only the remaining members, $\mathbf{N} - \mathbf{R}$, be able to decrypt the rekeying messages to get the updated group key $k_g^{'}$. This is accomplished by sending $k_g^{'}$ encrypted by the

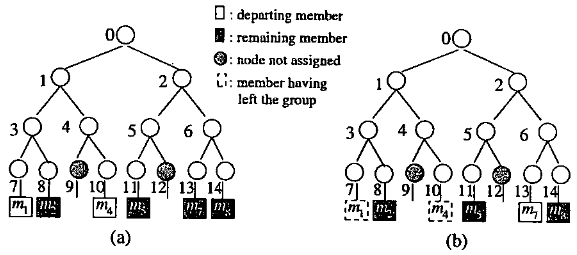

Figure 2: Stateless rekeying of SDR.

keys known only to $\mathbf{N}-\mathbf{R}$. To do so, $\mathbf{N}-\mathbf{R}$ is divided into a set of *subsets*, each of which associated with a key known only to the members covered by the subset. Then k_g' is encrypted by the keys of the resultant subsets and multicast.

Subsets are defined through the SDR key tree as follows [10]. For node u in the key tree, let S_u be the set of leaf nodes which are descendants of u. For two nodes u and v, where v is a descendant of u, we define subset $S_{u,v}$ as $S_u \backslash S_v$, i.e., $S_{u,v}$={leaf w | w is a descendant of u but not of v}. Using the partition algorithm in [10], $\mathbf{N}-\mathbf{R}$ can then be divided into total $SubN$ subsets: $\{S_{u_1,v_1}, S_{u_2,v_2}, \ldots, S_{u_{SubN},v_{SubN}}\}$ such that

$$\bigcup_{i=1}^{SubN} S_{u_i,v_i} = \mathbf{N}-\mathbf{R} \tag{1}$$

$$\text{and} \quad S_{u_i,v_i} \bigcap S_{u_k,v_k} = \emptyset \text{ if } i \neq k \tag{2}$$

. From (1) and (2), it is clear that any remaining member is exactly covered by one subset. Each subset $S_{u,v}$ is associated with a key $K_{u,v}$, which can be computed only by the members covered by $S_{u,v}$ using the information received during member registration.

SDR rekeying is then conducted by sending k_g' encrypted individually with keys K_{u_i,v_i} $(i = 1, \ldots, SubN)$. Based on (1), only the remaining members are able to decrypt the rekeying messages to get k_g'. For example, in Figure 2(a), when members m_1 and m_4 depart, the GCKS divides the remaining members into $S_{3,7} = \{m_2\}$ and $S_{2,12} = \{m_5, m_7, m_8\}$. The updated group key k_g' is encrypted by $K_{3,7}$ and $K_{2,12}$ separately and sent out, such that only m_2, m_5, m_7 and m_8 can retrieve k_g'. Figure 2 also shows the stateless property of SDR rekeying. Specifically, if member m_5 did not receive the rekeying message $\{k_g'\}_{K_{2,12}}$ sent when the members m_1 and m_4 departed, it is still able to decrypt the next rekeying message $\{k_g''\}_{K_{5,12}}$ sent when member m_7 departs later (Figure 2(b)), since $K_{5,12}$ is kept unchanged during rekeying.

Notice that the rekeying messages in SDR are also key-oriented, i.e., each message contains a single key (the updated group key) encrypted by the key of a subset.

In summary, Figures 1 and 2 demonstrate the distinction between LKH and SDR key trees for two consecutive rekeying instances in the same group. One can see that the LKH key tree can be dynamically adjust and has the varying n, whereas the SDR key tree has a fixed size of N. It should be noted that the relationship between N and n is application dependent. To get an idea for the comparison of N and n, consider that N would represent the total number of TVs tuned to *any* part of a SuperBowl broadcast, whereas n would represent the number of TVs tuned to *a given* part of the broadcast.

Having laid the necessary groundwork, we begin to compare the performance of the two rekeying algorithms, detailed in the next section.

3 Analytical comparison

Two performance metrics are considered in our comparison between LKH and SDR: (i) key storage at both member side and the GCKS side and (ii) rekeying cost. The rekeying cost includes unicast cost (to new joining members) and multicast cost (to other members).

As already mentioned, the performance of LKH depends on the height of the key tree, which is related to the degree of the key tree. Throughout our comparison, we use binary key tree for LKH considering that SDR is also based on binary key trees. LKH with other degrees have qualitatively similar results with the one with degree 2 as we will see from Figure 9 in Section 4.2.2.

3.1 Member and GCKS storage

We first compare the key storage in this subsection, followed by the rekeying cost in the next.

The GCKS using LKH needs to store all the keys in the key tree (internal nodes and leaf nodes), incurring a storage cost of $2n - 1$. Each member stores all the keys along its path to the root, i.e., $\log n$ keys.

The stateless property of SDR requires additional key storage at both the GCKS and the member. Specifically, the GCKS randomly chooses a key k_u associated with node u in the SDR key tree and selects a pseudo-random generator G, which extends a b-bit input x into a $3b$-bit output: $G : \{0,1\}^b \mapsto \{0,1\}^{3b}$. Let $G_L(x)$, $G_R(x)$ and $G_M(x)$ denote the left third, right third and middle third of the output of $G(x)$ respectively. To assign the keys of subsets, the GCKS also introduces a LABEL [10]. In particular, any node v has a LABEL to each of its ancestors u, denoted as $L_{u,v}$. For example, in Figure 2(a), node 4 has two labels: $L_{0,4}$ and $L_{1,4}$ to its two ancestors 0 and 1 respectively. Figure 3 shows the process used by the GCKS to generate $L_{u,v}$ based on k_u and G.

```
ComputeLabel ($u, v, k_u$) :
//Precondition: $v$ is a descendant of $u$

$L = k_u$ ;
while ($v$ is a descendant of $u$) {
    if ($v$ is a descendant of $u$'s left child)
        $L = G_L(L), u = u$'s left child
    else $L = G_R(L), u = u$'s right child
}
return $L$
```

Figure 3: Pseudo code for calculating LABEL $L_{u,v}$ on k_u [10].

Specifi cally, the key $K_{u,v}$ of the subset $S_{u,v}$ takes the value of $G_M(L_{u,v})$.

Based on the discussion above, the GCKS has to store one group key, $2N-1$ node keys and the keys of all possible subsets. To calculate the number of all possible subsets in an SDR key tree, we defi ne the *height* of a node in the key tree as the length of the path from the node to a leaf node and implicitly, the height of a leaf node is 0. The height of a key tree is consequently defi ned as the height of the root. Given an SDR key tree with height $h = \log N$, node u with height i has $2^{i+1}-2$ descendants, each of which, v, incurs a possible subset $S_{u,v}$. There are 2^{h-i} nodes with height i. Thus the number of all possible subsets is $\sum_{i=1}^{h} 2^{h-i}(2^{i+1}-2)$. This results in the key storage of the GCKS conducting SDR is

$$1 + (2N-1) + \sum_{i=1}^{h} 2^{h-i}(2^{i+1}-2) = 2N\log N + 2 \tag{3}$$

However, one might argue that the GCKS conducting SDR only needs to store a secret seed, and other keys, including group keys, node keys and keys of subsets, can be computed using one-way functions based on the seed. Particularly, the current group key can be the output of using a one-way function on the last one; the node keys can be computed based on the seed and the node ID; the keys of subsets can be obtained by executing the procedure in Figure 3 and applying G. Similarly, the GCKS conducting LKH only needs to store a secret seed based on this argument.

Although such argument sounds reasonable, we would like to point out that, in this case, the GCKS has to introduce much computation when rekeying. It is a trade-off between computation and storage. The summary of the key storage for the two schemes in Table 1 reflects this argument.

We have mentioned previously that the key of a subset can be computed by the members covered by the subset using the information received during member registration. Let I_A be the secret information that member A receives when registers. I_A includes the a set of LABELs necessary to obtain the keys of the subsets to which A may belong. The total number of subsets to which a member belongs is $O(N)$. However, due to two facts that a member belonging to subset $S_{u,v}$ always belongs to $S_{u,w}$ where w is a descendant node of v, and that $K_{u,w}$ for any descendant w of v can be computed if u, v, and $L_{u,v}$ is known, member A only needs to store the smaller set of LABELs: $\{L_{u,v}\}$ with size $(\log^2 N + \log N)/2$ [10], where u is an ancestor of A and v is the fi rst descendant of u off the path from A to u. To exemplify, consider the SDR key tree in Figure 2, member m_5 (node 11) belongs to 16 subsets, i.e., $S_{0,1}$, $S_{0,3}$, $S_{0,4}$, $S_{0,7}$, $S_{0,8}$, $S_{0,9}$, $S_{0,10}$, $S_{0,6}$, $S_{0,13}$, $S_{0,14}$, $S_{0,12}$, $S_{2,6}$, $S_{2,13}$, $S_{2,14}$,$S_{2,12}$, and $S_{5,12}$. However, m_5 only needs $I_{m_5} = \{L_{0,1}, L_{0,6}, L_{0,12}, L_{2,6}, L_{2,12}, L_{5,12}\}$ to calculate the keys for all of the 16 subsets. In [10], the authors point out that I_A also includes an additional key corresponding to the case when all potential members are in the group, i.e., $\mathbf{R} = \emptyset$. Then $|I_A| = (\log^2 N + \log N)/2 + 1$. Considering that each member also needs to store the group key, the key storage at member side is totally $(\log^2 N + \log N)/2 + 2$.

Table 1 summaries the above comparison, where we can see that SDR is expensive in both the member storage and the GCKS storage, due to the large key tree size ($N > n$).

Table 1: Key storage of LKH and SDR.

	GCKS storage	Member storage
LKH	$2n - 1$ or $1^{[1]}$	$\log n$
SDR	$2N \log N + 2$ or $1^{[1]}$	$(\log^2 N + \log N)/2 + 2$

*[1]: In the case that only the secret seed is stored.

It will be helpful to see the actual key storage requirements for GCKS and group members in practice. We list them in the next section when we have values for N and n in specifi c scenarios.

3.2 Rekeying cost

In this subsection, we compare the rekeying cost of LKH and SDR, including unicast cost and multicast cost.

Unicast is conducted to deliver information to a single receiver while multicast is applied to distribute for multiple receivers. Generally in rekeying, the GCKS unicasts the keys required to be known to the joining members, and multicasts the messages containing updated keys to other members in the group. Consequently, we separate the rekeying cost comparison into unicast cost and multicast cost. The unicast cost is then equal to the member's key storage shown in Table 1 and multicast cost is measured as the number of key-oriented multicast messages.

However, it should be pointed out that the unicast cost to a joining member in SDR varies, depending on whether the joining member is a new member or a rejoining member. This variance comes from the fact that the keys associated with the SDR key tree nodes are unchanged during rekeying. A leaf node can only be assigned to at most one member in SDR; when a member goes offline and joins the group again, the member is typically assigned the same leaf node as before such that the SDR key tree has size equal to the total number of unique members, rather than to the total number of joining instances. Thus if a joining member A had been in the group, the GCKS does not need to unicast any keys to A. If it is the first time that A joins the group, the unicast cost is the same as the key storage. But in LKH, every joining member, no matter new or rejoining, incurs the same amount of unicast cost equal to the key storage at member side.

Work in [6, 15] provides an analysis of the multicast cost for LKH rekeying. In LKH immediate rekeying, only the keys along the path from the departing/joining member to the root are updated. Each of the updated key is multicast encrypted by the KEKs corresponding to the node's children, resulting in $2 \log n$ multicast cost. Batch rekeying cost using LKH depends on the relationship between the number of joining members (j) and the number of departing members (r) during a batch. More details can be found in [6].

The multicast cost of SDR is equal to the number of resultant subsets for $\mathbf{N} - \mathbf{R}$ since the updated group key is multicast encrypted using the keys of the resultant subsets. D. Naor *et al.* pointed out in [10] that the SDR multicast cost is $2R - 1$ in the worst case and $1.38R$ in average. This statement is not complete since it overlooks the relationship between N and R. Since subset $S_{u,v} = S_u \backslash S_v$ ={descendants of node u}-{descendants of node v}, at least one member in $\mathbf{N}$ and one member in $\mathbf{R}$ are required to generate a subset of $\mathbf{N} - \mathbf{R}$. If either R or $N - R$ is small, fewer subsets are generated to cover $\mathbf{N} - \mathbf{R}$. Consequently, the multicast cost should be determined by $\min(N - R, R)$, achieving the highest value when R equals to $N/2$.

The preceding comparison is summarized in Table 2, where we do not include the cost of batch rekeying due to the complicated expression for LKH batch multicast cost detailed in [6]. As for the batch SDR multicast cost, we conjecture that it is similar with the immediate one since SDR is stateless. Such conjecture is verified by our simulation described in the next section. We also use simulation to do more detailed comparison on the batch multicast cost of the two schemes.

As a final comment, we regard the multicast cost as the critical cost of the rekeying cost for the following two reasons. First, multicast accounts for most of the rekeying traffic. Second, multicast cost is more expensive than unicast cost with respect to the number of links that a message travels. In the rest of this

Table 2: Rekeying cost of LKH and SDR.

	Unicast cost	Immediate multicast cost
LKH	$\log n$	$2 \log n$
SDR	$0^{[1]}$ or $(\log^2 N + \log N)/2 + 2^{[2]}$	$O(\min(R, N - R))$

*[1]: to rejoining members. [2]: to new members.

paper, we do not explicitly distinguish multicast cost from rekeying cost.

4 Comparison using simulation

In this section, we first verify the relationship of the SDR rekeying cost on N and R. We then use both real-life data and simulated data to compare the rekeying cost of LKH and SDR in different scenarios, which particularly includes immediate rekeying, periodic batch rekeying and membership batch rekeying.

4.1 SDR rekeying cost with N and R

We assume a fixed $N = 4096$ and construct a balanced SDR key tree with 4096 leaf nodes. We then vary the number of the remaining members $(N - R)$ and calculate the rekeying cost, which is equal to the number $(SubN)$ of resultant subsets of the $N - R$ members. For a given $N - R$, we simulate 10 instances, in each of which the remaining members are randomly assigned the leaf nodes independently. The average number of $SubN$ is then calculated on 10 instances for the corresponding $N - R$. Figure 4 shows the result. The figure confirms our conjecture that if either R or $N - R$ is small, $SubN$ is also small; $SubN$ achieves the highest value when $N - R = R = N/2$.

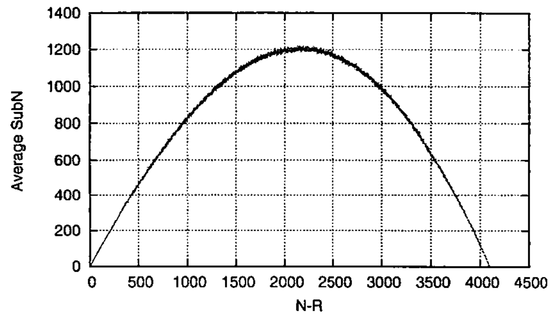

Figure 4: SDR rekeying cost with varying $N - R$ given N=4096.

4.2 Real-life MBone simulation

The real-life data comes from Multicast Backbone (MBone) information collected by Almeroth *et. al.* [1], including NASA STS-71, NASA STS-65, IMS, IPNG and UCB, total fi ve sessions. For a particular session, the collected data includes the *inter-arrival time* between two consecutive members joining the session and the *duration time* that a member staying in the session. Based on these data, we can obtain the membership. We then simulate rekeying on the membership change using LKH and SDR separately. Simulation results of the fi ve sessions were similar and we present the largest session, named STS-71.

According to [1], during the whole session, there are around 4000 unique hosts joined the STS-71 session. We then choose $N = 4096$ as the number of all potential members for this particular secure multicast session. When SDR is used, we construct a fi xed binary key tree with 4096 leaf nodes and randomly assign a leaf node to a new joining member. If a joining member had been in the group[2], the member is assigned the same leaf node as the one assigned before. If LKH is the rekeying mechanism, we dynamically adjust the binary key tree based on the algorithm in [4] to hold the currently active members in the multicast session.

We fi rst apply the immediate rekeying on STS-71 session, using LKH and SDR separately. Figure 5 shows the result, from which one can see that immediate rekeying cost using LKH is much smaller than the one using SDR. More specifi cally, the average LKH rekeying cost was 12.15 and the average height of the LKH key tree was 7.78 because of the dynamic adjustment. This result is consistent with Table 2. The average SDR rekeying cost was 138.98, which is comparable to the average number of the active members. The curves of the membership (n) and the SDR rekeying cost coincide in the fi gure. This is because R is very close to N in this scenario. The $N - R$ members staying in the group are dispersed in the key tree and most of resultant subsets only cover one remaining member. In this case, SDR rekeying cost $\approx N - R$.

Table 3: Average metrics of SDR and LKH with periodic batch rekeying in NASA STS-71 Session ($n_{max} = 329$).

T (mins)	$N - R$	$SubN$	$SubH$	$ADG_{\mathrm{N-R}}$	ADG_{R}	n	C_{LKH}	$TreeH$	j	r
20	93.90	92.65	1.02	1.03	2.07	99.74	31.17	7.56	5.85	5.81
40	93.88	92.64	1.02	1.03	2.07	105.59	45.15	7.68	11.69	11.61
60	93.77	92.52	1.02	1.03	2.07	111.34	54.7	7.68	17.52	17.40
120	93.87	92.63	1.02	1.03	2.07	129.09	76.96	7.41	34.95	34.72
240	93.58	92.28	1.02	1.03	2.07	164.35	98.65	7.68	69.88	69.37

[2]For privacy, the data we get does not have the identities of the hosts. We randomly assign each host an ID such that we can recognize the rejoining according to the ID.

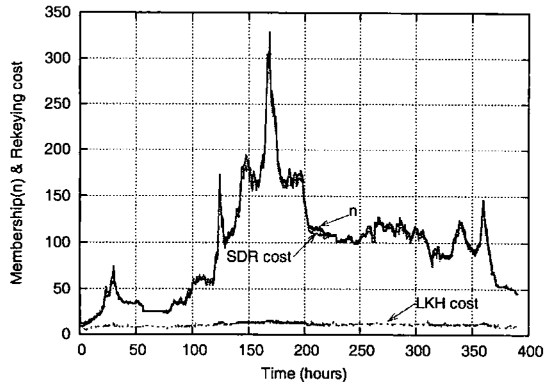

Figure 5: Rekeying cost of LKH and SDR for immediate rekeying in NASA STS-71 Session.

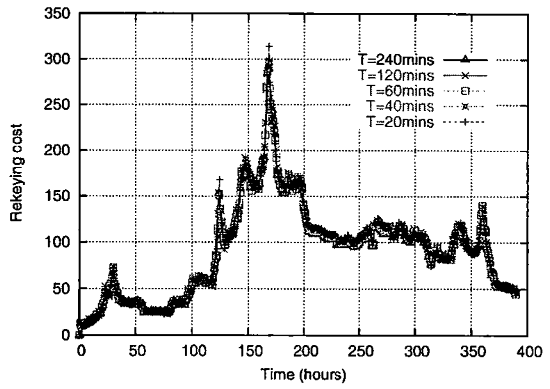

Figure 6: SDR rekeying cost for periodic batch rekeying in NASA STS-71 Session.

Figures 6 and 7 show the rekeying cost of SDR and LKH in periodic batch rekeying. Each fi gure includes the costs for fi ve different batch periods (T). From the fi gures, we observe that SDR rekeying cost remains almost the same in the fi ve different batch periods, while LKH rekeying cost increases as T increases.

Batch rekeying was proposed to reduce the rekeying cost [12]. To clear the confusion that in LKH, batch rekeying cost (Figure 7) looks much higher than immediate rekeying cost (Figure 5), it should be noted that the batch rekeying cost is the cost for the corresponding batch period, whereas, the immediate rekeying cost is instantaneous. If we add up all the immediate rekeying cost associated in a batch period, the sum is higher than the corresponding batch cost.

4.2.1 Explanation

To better understand the above results, we introduce the following metrics of the two schemes.

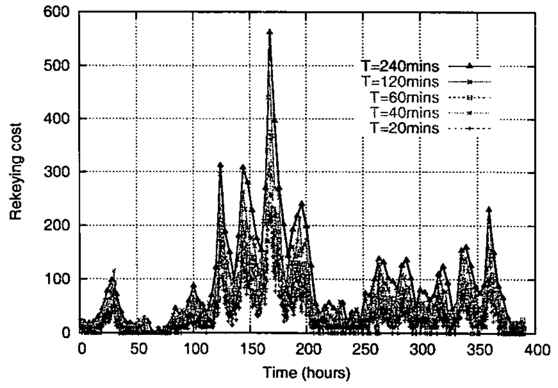

Figure 7: LKH rekeying cost for periodic batch rekeying in NASA STS-71 Session.

We use the height of node u in the key tree to represent the height of a subset $S_{u,v}$, denoted as $SubH_{u,v}$. Generally speaking, the higher a subset, the more members in $\mathbf{N}-\mathbf{R}$ it covers. For example, a subset $S_{u,v}$ with height 1 only includes one remaining member while excluding one member in $\mathbf{R}$, as in $S_{3,7}=\{m_2\}$ in Figure 2(a). We use $SubH$ to denote the average value of the heights of $SubN$ resultant subsets for $\mathbf{N}-\mathbf{R}$ in any rekeying instances. Thus, given a $N-R$, a larger $SubH$ generally implies a smaller $SubN$, and vice versa.

We also define *adjacent_degree* to denote the adjacency of the leaf nodes. Consider the leaf nodes in the SDR key tree in any rekeying instance. There are two kinds of leaf nodes, corresponding to members in $\mathbf{N}-\mathbf{R}$ and $\mathbf{R}$ respectively. Adjacent nodes of the same kind, corresponding to $\mathbf{N}-\mathbf{R}$ or $\mathbf{R}$, form a *block.* The leaf nodes are thus composed of alternate blocks. A block is associated with an adjacent_degree, defined as the length of the block, i.e., the number of nodes in the block. Note that for a block with a single node, the corresponding adjacent_degree is 1. $ADG_{\mathbf{N}-\mathbf{R}}$ is then defined as the average adjacent_degree of the blocks consisting of nodes corresponding to members in $\mathbf{N}-\mathbf{R}$. Similarly, we have the definition of $ADG_{\mathbf{R}}$. For example, the leaf nodes in the SDR key tree in Figure 8 form 4 blocks of $\mathbf{N}-\mathbf{R}$ and 4 blocks of $\mathbf{R}$, incurring 5 resultant subset: $S_{1,4}, S_{9,19}, S_{10,21}, S_{5,12}$ and $S_{13,28}$. The adjacent_degrees of the four blocks of $\mathbf{N}-\mathbf{R}$ are 4, 1, 3 and 1, corresponding to blocks $\{m_1, m_2, m_3, m_4\}$, $\{m_6\}$, $\{m_8, m_9, m_{10}\}$ and $\{m_{13}\}$, respectively, resulting in $ADG_{\mathbf{N}-\mathbf{R}}=2.25$. $ADG_{\mathbf{R}}=1.75$ in this example as the four adjacent_degrees of blocks of $\mathbf{R}$ are 1, 1, 2 and 3. Generally, a small $ADG_{\mathbf{N}-\mathbf{R}}$ implies that the remaining members are dispersed in the key tree and normally, more subsets are generated to cover $\mathbf{N}-\mathbf{R}$. On the other hand, a larger $ADG_{\mathbf{N}-\mathbf{R}}$ indicates that the remaining members may be grouped, incurring fewer subsets.

Table 3 shows the average metrics of the periodic batch rekeying in NASA STS-71 Session. Both $SubH$

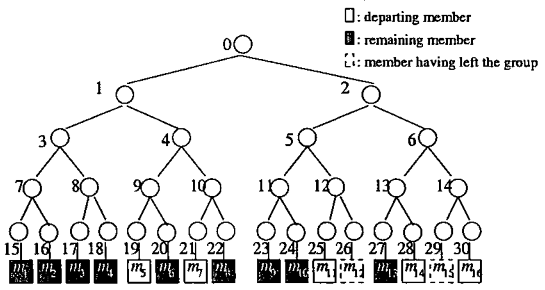

Figure 8: An SDR key tree.

and $ADG_{\mathrm{N-R}}$ are close to 1, indicating that most subsets contain only one remaining member. This is reasonable since around 300 remaining members are likely to be dispersed in the key tree with 4096 leaf nodes. In Table 3, C_{LKH} is the rekeying cost using LKH. $TreeH$ is the average value of the heights of the LKH key tree conducting dynamic contraction. j and r are the number of joining members and departing members during a batch period T, respectively. From the result in [6], LKH rekeying cost increases as j and r increase.

From Figures 5, 6, 7 and Table 3, we can see that LKH performs better in immediate rekeying ($j = 1$ or $r = 1$) and small batch rekeying (T is small), but as T increases, SDR rekeying begins to perform better than LKH. When T=240 minutes, the average LKH rekeying cost is higher than that in SDR. Finally, we observe that LKH rekeying cost is sensitive to T while SDR cost is insensitive. Such results are more obvious if we conduct membership batch rekeying shown next.

4.2.2 Cross-over point in membership batch rekeying

When membership batch rekeying is conducted in NASA STS-71 session, the GCKS performs rekeying when it receives a predefi ned number of departing requests. The threshold is then equal to r, the number of departing members since last rekeying. We vary the threshold r from 5 to 100. For a given r, membership batch rekeying with the particular r is conducted throughout the whole STS-71 session, and then the average rekeying cost is calculated for the corresponding r. Figure 9 presents the result. It is interesting to observe from the fi gure that the LKH rekeying cost monotonously increases with r, whereas the SDR rekeying cost keeps stable, demonstrating a *cross-over point* of LKH to SDR graphs. Thus it is clear from the result that LKH performs better for immediate rekeying as well as small batch rekeying whereas SDR rekeying is better for large batch rekeying. Note that the cross-over point of LKH degree=2 to SDR is for a batch size $r = 55$, whereas the maximum group size was 329 and the average group size was 100. We also observe from Figure 9 that LKH achieves least rekeying cost with degree=4. This has been reported earlier in the literature [15].

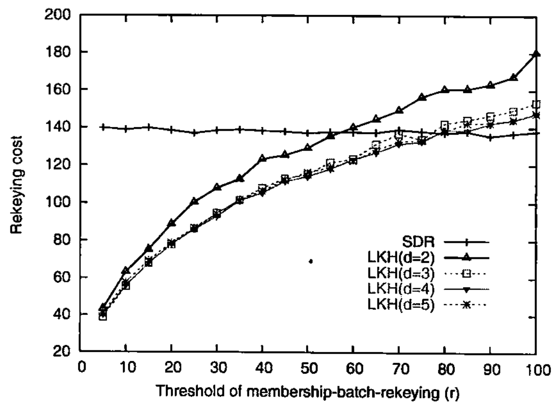

Figure 9: LKH and SDR rekeying cost for membership batch rekeying in NASA STS-71 Session.

4.2.3 Actual key storage

Finally, we calculate the key storage requirements of LKH and SDR in STS-71 session. Considering 3DES for encryption, we need 128-bit or 16B keys. Due to the fact that in LKH, key storage at the GCKS and group members changes with n, we compute the average values. Using the results in Table 1, the average key storage is 90.83 bytes at member side and 3002.90 bytes at the GCKS side when the GCKS stores the keys rather than the secret seed. In SDR, both GCKS and group members have fi xed size key storage, which are 1572864 bytes and 1232 bytes respectively given that $N = 4096$. Thus we can see that stateless schemes require much more key storage than in LKH, at both the GCKS and the group members.

4.3 DaSSF simulation

As we have seen in the NASA STS-71 session, the number of active members, n, is much smaller than the number of potential members, N. It is also desirable to compare the performance of LKH and SDR in the group where n is more signifi cant. In this subsection, we use DaSSF [5] to generate such groups.

By DaSSF, we can simulate members' activities. Particularly, in the model we consider, the group is empty initially. Each potential member waits for a randomly distributed waiting time, W, before he/she joins the group, and consequently stays in the group for another randomly distributed duration time, D, before he/she leaves the group. All potential members repeat this process till the end of the multicast session. By specifying different distributions of W and D, we can obtain different group behaviors.

We first use DaSSF to further explore the effect of the member adjacency to the SDR rekeying cost, and then compare the rekeying cost of LKH and SDR in the simulated group with larger n comparing to N.

4.3.1 Effect of member adjacency

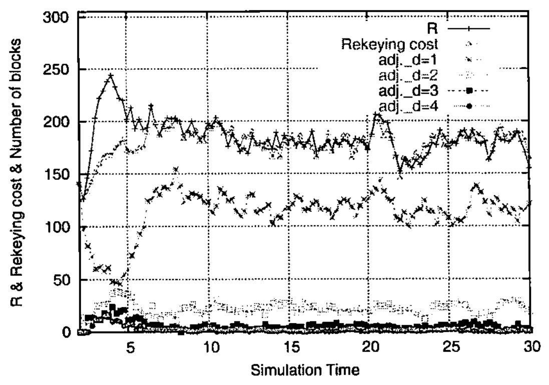

Figure 10: SDR rekeying cost and average adjacent_degree.

We simulate a group with $N = 1024$, where W is exponentially distributed with mean 1, i.e., each member waits for an average time of 1 unit of simulation time before he/she joins the group, and D is a distribution of Pareto(2,1.6). We then conduct SDR periodic batch rekeying on this group with $T = 0.25$ unit of simulation time. Figure 10 shows that the SDR rekeying cost mostly coincides with R. This is because $R \approx 200$, much smaller than $N - R$. At any rekeying instance, we also calculate the number of the blocks of nodes corresponding to members in **R** with 4 different adjacent_degrees: 1, 2, 3 and 4[3], which is also shown in Figure 10. From the figure, we observe that rekeying cost reduces if the number of blocks with large adjacent_degree increases. In particular, from simulation time 0 to 8, the number of blocks with adjacent_degree 3 or 4 is relatively much larger than the ones in the rest of simulation period, and the number of blocks with adjacent_degree 1 is relatively small. The larger number of blocks with adjacent_degree 3 or 4 in this period indicates that revoked members are more adjacent, incurring smaller rekeying cost. In the rest of the simulation period from time 8 to 30, blocks with adjacent_degree 1 dominates all the blocks, indicating that members in **R** are mostly separate. As a consequence, each resultant subset only excludes one member in **R** and the rekeying cost is very close to R.

Generally speaking, when members in $\mathbf{N} - \mathbf{R}$ (or **R**) are more adjacent, the number of resultant subsets

[3]Blocks with adjacent_degree greater than 4 is rare and we do not include in the figure.

and then the SDR rekeying cost is reduced.

4.3.2 Large simulated group behavior

SDR is not efficient at all in NASA STS-71 session as we have seen in the previous section if we consider the ratio of the SDR rekeying cost ($SubN$) to the number of remaining members ($N-R$), i.e., $SubN/(N-R) \approx 1$. In that case, SDR rekeying was as expensive as encrypting the updated group key separately for each active member. That is mainly due to $N - R$ being much smaller than N. We notice that in some real applications, $N - R$ can be much larger than R. For instance, consider the audience of pay-per-view events, or a football game. Most people join at some point during the game and many (around half the audience) tune in from start to finish. During the session, many people join and leave, while most start leaving towards the last quarter of the lifetime of the group. Such groups are larger in size, than the NASA STS-71 session. This motivates us to simulate a larger group using DaSSF.

We assume a larger size of potential members, $N = 64K$. Then by adjusting members' waiting time (W) and duration time (D) in DaSSF, we simulated a group with membership (n) shown as the top curve in Figure 13. Notice that n is much larger compared to the STS-71 session, and more importantly it is closer to N than in that session. This membership has a peak (almost all the potential members are in the group) at simulation time 15 and the average number of active members through the whole session exceeds half of N. This simulated membership provides us a group that behaves differently than the STS-71 session. We investigate the relative advantages of LKH and SDR rekeying for this group.

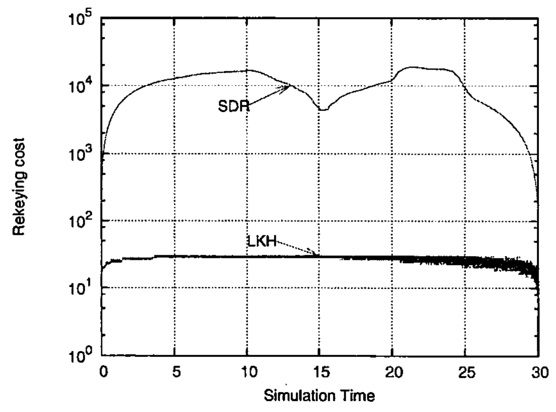

Figure 11: Rekeying cost of SDR and LKH for immediate rekeying in simulated group with $N = 64K$.

Figure 11 shows the immediate rekeying cost of SDR and LKH in the simulated group. Compared to the cost of SDR, LKH immediate rekeying is much less so we plot the rekeying cost in log-scale. Similar to Figure 5, LKH rekeying cost for immediate rekeying is related with the height of the key tree. Whereas, SDR rekeying cost is much different with the one in Figure 5. To be clear, we plot the SDR cost separately in Figure 12, where we include the curve of R to show that SDR rekeying cost is related to N and R. It is interesting to note that although $N - R$ increases in period from simulation time 10 to 15, SDR rekeying cost decreases. The reason is that R decreases in this period. A subset $S_{u,v}$ not only needs to cover members in $\mathbf{N} - \mathbf{R}$ (descendants of node u but not of v), but also excludes some members in $\mathbf{R}$ (descendants of node v). So fewer subsets are required when R decreases. This result also confirms our conjecture that SDR rekeying cost is determined by $\min(N - R, R)$.

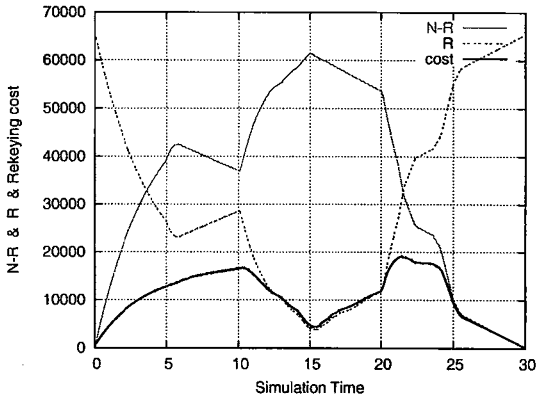

Figure 12: SDR rekeying cost for immediate rekeying in simulated group with $N = 64K$.

We plot the batch rekeying cost of LKH and SDR in the simulated session in Figure 13 and 14 respectively, each of which includes the costs of five different batch periods (T). Again, while LKH rekeying cost increases as T increases, SDR rekeying cost stays nearly the same for different T, much less than $N - R$. Numerically, $SubN/(N - R) \approx 1/3$. Thus compared to STS-71 session, SDR rekeying is more efficient in the group with larger $N - R$. We also observe that in this simulated group, SDR rekeying cost for immediate rekeying is almost the same as the one for batch rekeying, which is also observed in STS-71 session (Figure 5 and 6). The reason is the stateless property of SDR. More specifically, the SDR rekeying cost at a given point of time only depends on the positions of members in $\mathbf{N} - \mathbf{R}$ in the SDR key tree at that time, no matter what rekeying mechanism is used, e.g., immediate rekeying or batch rekeying. Thus in terms of rekeying cost normalized by time, SDR performs more efficient as the batch size increases. However, the larger batch size is, the more information exposed to unauthorized members. This issue is addressed in the exposure-oriented rekeying [17].

Table 4 shows the average metrics for the two schemes, which allows us to take a closer look at how these algorithms perform. There is also a cross over point where SDR begins to perform better than LKH,

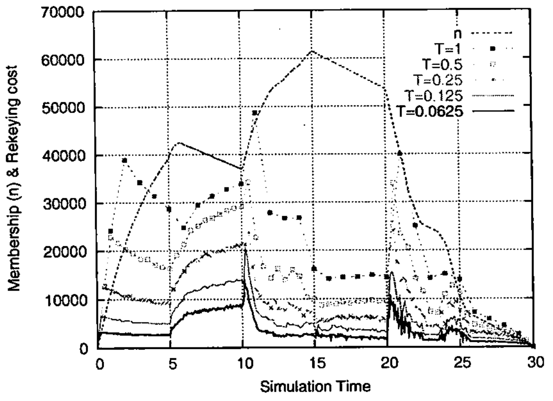

Figure 13: LKH rekeying cost for batch rekeying in simulated group with $N = 64K$.

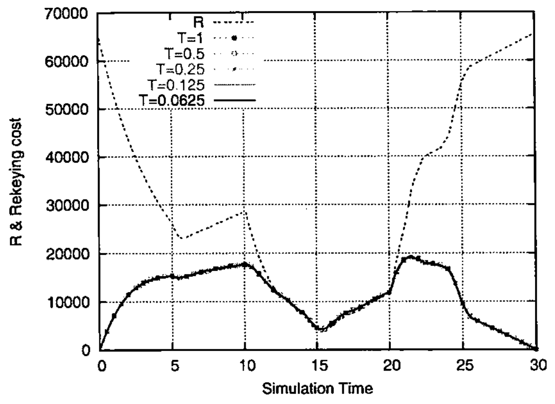

Figure 14: SDR rekeying cost for batch rekeying in simulated group with $N = 64K$.

Table 4: Average metrics of SDR and LKH with periodic batch rekeying in DaSSF simulation with $N = 64K$ ($n_{max} = 61806$).

T[1]	$N - R$	$SubN$	$ADG_{\mathbf{N-R}}$	$ADG_{\mathbf{R}}$	n	C_{LKH}	$TreeH$	$j = r$[2]
0.0625	34514.52	11226.77	4.13	1.28	34929.22	3506.51	15.63	369.48
0.125	34441.98	11227.00	4.11	1.27	35249.88	5932.27	15.65	738.97
0.25	34585.75	11256.74	4.13	1.27	36117.34	9548.78	15.66	1477.93
0.5	34003.56	11092.56	4.04	1.24	36800.26	14868.53	15.70	2955.87
1	34550.70	11243.27	4.10	1.24	39467.87	22008.26	15.73	5911.73

*[1]: Units of the batch period compared to the 30-unit of total simulation time.

[2]: The simulation starts with $n = 0$ and ends with $n = 0$, thus the total number of joining members equals to the total number of departing members during the whole simulation.

in the simulated group. Note that for batch period $T = 0.5$, LKH rekeying cost is higher than that in SDR.

4.3.3 Actual key storage

Perhaps unsurprisingly, the large storage cost becomes more pronounced in the larger group. LKH key storage required in the simulation scenario for GCKS and members are $1030K$ bytes and 228 bytes respectively. Whereas, the key storage of GCKS and group members in SDR are fi xed to $32768K$ bytes and 2176 bytes respectively, given $N = 64K$.

4.4 Summary of the simulation

The simulation results based on both MBone STS-71 session and DaSSF simulated group show that SDR rekeying cost is much higher than LKH cost in immediate rekeying and small batch rekeying, whereas SDR outperforms LKH when the batch rekeying period increases.

Since each member in **N-R** is covered by one and exactly one subset, it is obvious that the number of resultant subset $SubN \leq N - R$. However, $SubN$ may exceed R when $R \leq N - R$. For example, consider the subtree rooted at node 1 in Figure 8, where $R = 2$, $N - R$=6, and the number of subsets incurred by this subtree is 3, e.g., $S_{1,4}$, $S_{9,19}$ and $S_{10,21}$. This is also the worst case of $SubN$ where $SubN = 2R - 1$ [10].

Proposition 1. *If $SubN > R$, the SDR must include at least one subtree $\mathcal{T}_s$ that satisfies the following three conditions. Let u be the root of $\mathcal{T}_s$.*

(1) u has a child v_1. All the members attached to the descendants of v_1 are in **N-R***;*

(2) Among all the members attached to the descendants of the other child v_2 of u, at least two but not all are in **R***;*

(3) Any two members in **R** *attached to the descendants of v_2 are NOT siblings.*

Proof: Our proof applies the induction on the height of the SDR key tree.

Basic case: Consider an SDR key tree $\mathcal{T}$ with height of 3, it is not hard to enumerate that the only case where $SubN > R$ is when $R = 2$ and $SubN = 3$. The subtree rooted at node 1 in Figure 8 shows this case. The subtree in the fi gure satisfi es the three conditions.

Inductive case: Let us assume that all SDR key trees $\mathcal{T}$ with height $\leq h$ satisfy proposition 1. Consider a tree $\mathcal{T}$ with height $h + 1$ and $SubN > R$ in $\mathcal{T}$. Let u be the root of $\mathcal{T}$. Let v_1 and v_2 be two children of u. Denote the subtree rooted with v_1 and v_2 as $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively.

If both $\mathcal{T}_1$ and $\mathcal{T}_2$ have some leaf nodes attached by members in $\mathbf{R}$, the number of subsets in $\mathcal{T}$ is the sum of the number of the subsets in $\mathcal{T}_1$ and $\mathcal{T}_2$. Only two possible subsets S_{u,v_1} and S_{u,v_2} are not in $\mathcal{T}_1$ nor $\mathcal{T}_2$. But these two subsets can not be generated if both $\mathcal{T}_1$ and $\mathcal{T}_2$ have some leaf nodes in $\mathbf{R}$. In this case, we are done since the height of both $\mathcal{T}_1$ and $\mathcal{T}_2$ is h.

Otherwise, without loss of generality, let all leaf nodes of $\mathcal{T}_1$ be attached by members in $\mathbf{N} - \mathbf{R}$, then all R members in $\mathbf{R}$ are in $\mathcal{T}_2$. There exist three cases:

(a) The number of subsets in $\mathcal{T}_2$, $SubN_{\mathcal{T}_2} < R$. This is not true since a single subset S_{u,v_2} will cover all leaf nodes of $\mathcal{T}_1$. The total number of subsets in $\mathcal{T}$ is then equal to $SubN_{\mathcal{T}_2} + 1 \leq R$.

(b) $SubN_{\mathcal{T}_2} > R$. According to the inductive assumption, $\mathcal{T}_2$ has height h and thus includes such subtree $\mathcal{T}_s$.

(c) $SubN_{\mathcal{T}_2} = R$.

(i) If $\mathcal{T}_2$ has only one leaf node, denoted as v, attached by a member in $\mathbf{R}$ (i.e., R=1), a single subset $S_{u,v}$ then covers all members in $\mathbf{N} - \mathbf{R}$ in $\mathcal{T}$. In this case $SubN = R = 1$.

(ii) It is not the case that $\mathcal{T}_2$ has all leaf nodes attached by members in $\mathbf{R}$. Otherwise, a single subset S_{u,v_2}, is enough to cover all members in $\mathbf{N} - \mathbf{R}$, i.e., $SubN < R$.

(iii) Now $\mathcal{T}_2$ has at least two but not all leaf nodes in $\mathbf{R}$. Assume that $\mathcal{T}_2$ has two leaf nodes: w_1 and w_2, attached by members in $\mathbf{R}$, and w_1 and w_2 are siblings. Since w_1 and w_2 are siblings, they are excluded by one resultant subset or they are not excluded by any resultant subset[4]. In both of cases, the rest of $R - 2$ members at least generate $R - 1$ subsets. Then there must be a subtree $\mathcal{T}_s$ of $\mathcal{T}_2$ which generates more subsets than the numbers in $\mathbf{R}$ attaching to leaf nodes in $\mathcal{T}_s$.

(iv) The last case is that $\mathcal{T}_2$ has no leaf nodes attached by members in $\mathbf{R}$ which are siblings. In this case, $\mathcal{T}$ itself is such a tree satisfying the three above conditions, which completes our induction. ■

Note that in such a subtree $\mathcal{T}$, the number of subsets incurred by $\mathcal{T}$ is 1 more than the number of leaf nodes of $\mathcal{T}$ corresponding to $\mathbf{R}$.

In Figure 11, we can see that from simulation time 13 to 20, $SubN$ is larger than R. We calculate the number of the subtrees $\mathcal{T}$ satisfying the above conditions, denote as τ, and plot it in Figure 15. We can see that τ is much higher in the period from simulation time 13 to 20 than in other periods.

[4]Equations 1 and 2 only guarantee that all members in $\mathbf{N} - \mathbf{R}$ are covered by resultant subsets, but some members in $\mathbf{R}$ may not be excluded by any result subset, like the m_{15}, m_{16} in Figure 8.

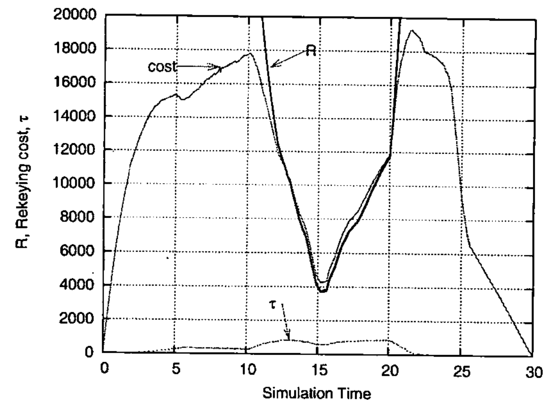

Figure 15: Relationship of R, rekeying cost and τ in the simulated group with $N = 64K$.

Then we observe that SDR rekeying cost (number of subsets) approximately equals to

$$\begin{cases} \frac{N-R}{ADG_{\mathbf{N-R}}}, & \text{if } N-R \leq R \\ \frac{R}{ADG_{\mathbf{R}}} + \tau, & \text{if } N-R > R \end{cases} \tag{4}$$

From Equation (4), it follows that the adjacency of members in $\mathbf{N} - \mathbf{R}$ or $\mathbf{R}$ directly affects rekeying cost in SDR. In our simulation, we assigned members to positions in the key tree randomly. In real-world deployments it may be more effi cient to assign members based on their join and departure behavior. For example, note that members from a timezone may join and leave a conference call around the same time. Therefore, SDR rekeying would be effi cient if all members within a timezone (or from an offi ce location) are assigned to neighboring positions within the key tree.

Finally, the SDR rekeying cost at a given point of time is independent on the rekeying mechanisms, e.g., immediate rekeying or batch rekeying. It only depends on the positions of members in $\mathbf{N} - \mathbf{R}$ in the SDR key tree at that time. The positions of members in $\mathbf{N} - \mathbf{R}$ in the SDR key tree in a rekeying instance affect the SDR rekeying cost through the adjacency, as Equation (4) states.

5 Conclusion and future work

In this paper, we use simulation to verify analytical comparison between logic key hierarchy based group rekeying scheme and a stateless scheme, *SDR*. Our simulation shows that LKH outperforms SDR in immediate rekeying and small batch rekeying. We also observed that LKH is preferred in the scenarios where the number of current members $n \ll N$, and membership variance is low. In groups where $n \approx N$, SDR performs better for batch rekeying. If $n \ll R$, subset rekeying cost coincides with n, i.e., each of the remaining members belongs to a different subset. Although the average SDR rekeying cost is $1.3R$ in [10], it may be much less than R in most cases, because of the member adjacency factor.

We did not address packet loss in this paper, which might affect the performance of the rekeying algorithms. Our next course of work is to consider this issue. Furthermore, the result shown in Equation (4) comes from our simulations. Mathematics analysis of this result might be another future work.

References

[1] K. C. Almeroth and M. H. Ammar. "Collecting and modeling the join/leave behavior of multicast group members in the MBone". In *Proceedings of the Symposium on High Performance Distributed Computing*, pp.209-16,Syracuse, NY, August 1996.

[2] I. Chang, R. Engel, D. Kandlur, D. Pendrakis, and D. Saha. "Key management for secure Internet multicast using boolean function minimization techniques". In *IEEE INFOCOM'99*, volume 2, pp. 689-698, New York, March 1999.

[3] W. Chen and L. R. Dondeti. "Performance comparison of stateful and stateless group rekeying algorithms". Technical Report 03-xx, Dept. of Computer Science, University of Massachusetts, Amherst, 2003.

[4] L. R. Dondeti, S. Mukherjee, and A. Samal. "DISEC: A distributed framework for scalable secure many-to-many communication". In *Fifth IEEE Symposium on Computers and Communications*, Antibes-Juan les Pins, France, July 3-6, 2000.

[5] Dartmouth Scalable Simulation Framework. `http://www.cs.dartmouth.edu/~jasonliu/projects/ss`

[6] X. Li, Y. Yang, M. Gouda, and S. Lam. "Batch rekeying for secure group communications". In *Proceedings of World Wide Web Conference 10 (WWW10)*, Hong Kong, May 2001.

[7] D. A. McGrew, and A. T. Sherman. "Key establishment in large dynamic groups using one-way function trees". Technical Report No. 0755, TIS Labs at Network Associates, Inc., Glenwood, MD, May, 1998.

[8] S. Mittra. "Iolus : A framework for scalable secure multicasting". In *ACM SIGCOMM'97*, pp.277-288, Cannes, France, September 1997.

[9] M. J. Moyer, J. R. Rao and P. Rohatgi. "A survey of security issues in multicast communications". *IEEE Network Magazine*,November/December 1999.

[10] D. Naor, M. Naor and J. Lotspiech. "Revocation and tracing schemes for stateless receivers". In *Advances in cryptology - CRYPTO'01*, Lecture Notes in Computer Science, volume 2139, pp.41-62, 2001

[11] S. Rafaeli. "A decentralised architecture for group key management". PhD appraisal, Computing Department, Lancaster University, Lancaster, UK, January 2000.

[12] S. Setia, S. Koussiah, S. Jajodia, and E. Harder. "Kronos: A scalable rekeying approach for secure multicast". In *Proceedings of IEEE Symposium on Security and Privacy*, Berkeley, CA, May 2000.

[13] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean. "Self-healing key distribution with revocation". In *Proceedings of IEEE Symposium on Security and Privacy*, The Claremont Resort Oakland, CA, May 2002.

[14] D. Wallner, E. Harder, and R. Agee. "Key management for multicast: Issues and architectures", IETF Informational RFC, June 1999. `http://www.faqs.org/rfcs/rfc2627.html.`

[15] C. K. Wong, M. G. Gouda, and S. S. Lam. "Secure group communications using key graphs". In *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 16–30, Feburary 2000.

[16] Y. R. Yang, X. S. Li, X. R. Zhang, and S. S. Lam. "Reliable Group Rekeying: A Performance Analysis". In *ACM SIGCOMM'01*, pp. 27-38, San Diego, CA, August 2001.

[17] Q. Zhang and K. L. Calvert. "On rekey policies for secure group applications". Presented to IRTF GSEC meeting at Vienna, Austria, July 2003, `http://protocols.netlab.uky.edu/~calvert/private/icccn03- submit.pdf.`