On Dynamic Subset Difference Revocation Scheme *

Weifeng Chen[1], Zihui Ge[2], Chun Zhang[1], Jim Kurose[1], Don Towsley[1]

[1] Department of Computer Science
University of Massachusetts, Amherst
{*chenwf, czhang, kurose, towsley*}*@cs.umass.edu*
[2] AT&T Labs-Research, Florham Park, NJ 07932
gezihui@research.att.com

Abstract

Subset Difference Revocation (SDR) [7] has been proposed to perform group rekeying in a stateless manner. However, statelessness comes at a cost in terms of key storage and messaging overhead when the number of currently active members is much smaller than the number of potential group members [3]. In this paper, we propose a *dynamic* SDR scheme to address these two problems. Briefly, rather than maintaining a large static key tree that can accommodate all potential group members, we use a smaller dynamic key tree for only currently active members. We dynamically assign current members to the positions in the key tree rather than using fixed pre-assignment. The smaller key tree requires less key storage and dynamic assignment achieves a smaller rekeying cost. We also describe enhancements to dynamic SDR that further improve performance. Our evaluation shows that the dynamic scheme significantly improves the performance of SDR, reducing by half the rekey communication cost in the case that the number of the currently active members is much less than the total number of potential members. Also, compared to the SDR in [7], dynamic SDR does not need to know the maximum number of potential group members in advance, a value that can be difficult to estimate in practice.

keywords: System design, Network security, Group rekeying, Subset Difference Revocation

*This research has been supported in part by the NSF under grant awards UF-EIES-0205003-UMA and EIA-0080119. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

Membership-based applications, such as pay-per-view and specialized information services (e.g., stock price, live news), require that information content be delivered to (and only to) subscribed members. This is typically accomplished by encrypting data using a common *Traffic Encryption Key* (TEK) that is shared by all currently active members. When a member joins the group, the TEK must be changed to ensure that the newly joining member cannot decrypt previous communications (a requirement known as "backward confidentiality"). Similarly, the TEK must be changed when a member leaves the group to ensure that future messages cannot be decrypted by the departing member (a requirement known as "forward confidentiality"). The algorithms that manage the distribution, updating and revocation of the TEK are collectively known as *group key management protocols*. The IETF MSEC framework suggests using a *Group Controller and Key Server (GCKS)* for rekeying. Generally, the TEK is encrypted using *Key Encryption Keys* (KEKs) and then multicast by the GCKS. Several works have dealt with the group rekeying problem [2, 5, 6, 7, 8, 9, 10].

Subset Difference Revocation (SDR) [7] has been proposed as a "stateless" group rekeying algorithm. By stateless, it is meant that members do not need to keep track of rekeying messages in order to maintain their states. A consequence of statelessness is that a member that misses one or more rekeying messages can still receive and decrypt future rekeying messages correctly, without having to first recover the missing keys. This very desirable property comes as a price, however [3] - SDR can require high key storage at both the member and GCKS sides, and can generate a significant amount of messaging traffic during rekeying.

These two problems arise from the fact that SDR maintains a static key tree that is constructed in advance. This tree must be large enough to hold all potential members. Since key storage cost is determined by the size of the key tree [7], key storage costs can thus be large. Further, a member joining the group is attached to a pre-assigned position in the key tree. Assuming that member activity is independent of position, when the number of currently active members is much smaller than the number of potential members, the positions occupied by active members are likely to be *sparse*, i.e., there are many holes (positions not occupied by an active member) among the positions occupied by active members. The sparse distribution of these positions can cause SDR to perform as inefficiently as encrypting the TEK separately for each active member [3]. We will refer to the SDR proposed in [7] as *static* SDR in this paper.

In this paper, we propose a *dynamic SDR* scheme to reduce both the key storage cost and the rekey communication cost of static SDR. Dynamic SDR uses a key tree that is large enough to hold currently active members (as opposed to *all* members, both active and inactive) in order to reduce key storage. This

is done by dynamically assigning a joining member a new position in the tree. Ideally, our goal will be to position active members adjacent in the key tree in order to reduce rekeying cost.

This paper has the following contributions. First, we design a new group rekeying algorithm, dynamic SDR. The algorithm is based on Subset Difference and uses a dynamic key tree to reduce both storage costs and messaging overhead. The algorithm is "multicast stateless" (i.e., it does not require nodes to maintain states when receiving the multicast rekeying messages, which distribute only TEKs) and does not require *a priori* knowledge of the number of potential members. Secondly, we propose and evaluate several enhancements on dynamic SDR. Our simulations show that dynamic SDR significantly reduces both the key storage cost and rekey communication cost when the number of currently active members is much less than the number of potential members. Finally, we investigate the tradeoff between the unicast and multicast costs in dynamic SDR.

The rest of the paper is organized as follows: In Section 2, we briefly overview the static Subset Difference Revocation algorithm. The dynamic SDR scheme is described in Section 3. Section 4 presents our evaluation. Section 5 discusses various properties of dynamic SDR. Related work is given in Section 6. Finally, we conclude the paper in Section 7.

2 Background: Subset Difference Revocation Algorithm

Static SDR [7] is a tree-based group key management protocol. Under static SDR, the GCKS maintains a binary key tree and assigns a fixed position (a leaf in the key tree) to each distinct member. For a finite set of potential members $\mathcal{N}$ ($N = |\mathcal{N}|$), it needs to maintain a tree with N leaves. For simplicity, we assume that $\log_2 N$ is an integer and that the key tree is balanced. Let $\mathcal{M} \subseteq \mathcal{N}$ be the set of members currently active in the group and $M = |\mathcal{M}|$. To distribute an updated TEK, the GCKS uses a *Subset-Cover* framework to partition $\mathcal{M}$ into disjoint subsets such that the union of the resultant subsets is $\mathcal{M}$, i.e., a member belongs to a resultant subset iff the member is in $\mathcal{M}$. Each subset is assigned a long-lived key during initialization. The updated TEK is encrypted with the keys of the resultant subsets individually and multicast to $\mathcal{M}$. Thus the keys of subsets are KEKs of SDR. SDR is a *stateless algorithm* for the KEKs are unchanged after initialization such that each member m in $\mathcal{M}$ is able to deduce the KEKs at each rekeying instance based on the *secret information* received during initialization. The secret information allows m to deduce the keys of *all* subsets to which m belongs.

In static SDR, the subsets are defined through the binary key tree. For a node i in the key tree, let S_i

```
ComputeLabel (i, j, L_i) :
//Precondition: j is a descendant of i
(1)   u = i, X = L_i ;
(2)   while (j is a descendant of u) {
(3)       if (j is a descendant of u's left child)
(4)           X = G_L(X), u = u's left child;
(5)       else X = G_R(X), u = u's right child;
(6)   }
(7)   return X;
```

Figure 1: Calculating $L_{i,j}$ on L_i [7]

be the set of the potential members which are descendants[1] of i. Given two nodes i and j, where j is a descendant of i, subset $S_{i,j}$ is defined as $S_i \backslash S_j$. Let $K_{i,j}$ be the key for $S_{i,j}$. $K_{i,j}$ can be computed as follows [7]. Consider the subtree T_i (rooted at node i) in the key tree. The root i is assigned by the GCKS a label L_i. Given that a parent was labeled X, its two children are labeled $G_L(X)$ and $G_R(X)$ respectively, where G is a pseudo-random sequence generator that outputs a random sequence whose length is three times the length of the input (i.e., $G : \{0,1\}^x \mapsto \{0,1\}^{3x}$); $G_L(X)$, $G_M(X)$ and $G_R(X)$ denote the left third, the middle third and the right third of the output of input X. Let $L_{i,j}$ be the label of node j derived in the subtree T_i from L_i, following such a labeling process (pseudo-coded in Fig. 1). Then $K_{i,j}$ takes the value of $G_M(L_{i,j})$.

A member m is able to compute the key of a subset iff m belongs to the subset. During initialization, each member m in $\mathcal{N}$ is delivered some secret information I_m to compute relevant KEKs ($K_{i,j}$). For a subtree T_i such that m is a leaf of T_i, m belongs to subset $S_{i,j}$ iff j is not an ancestor of m. Consider the path from i to m and let $i_1, i_2, \ldots, i_h$ be the nodes just "hanging off" the path, i.e., they are adjacent to the path but not ancestors of m [7] (see Fig. 2(a)). Each node j in T_i that is not an ancestor of m is a descendant of one of $\{i_1, i_2, \ldots, i_h\}$. Observe that m is able to compute $L_{i,j}$ for any descendant j of i_k if L_{i,i_k} is known to m. Thus in subtree T_i, m needs to know h labels: $\{L_{i,i_1}, L_{i,i_2}, \ldots, L_{i,i_h}\} \subset I_m$. There are total $\log N$ nodes that are ancestors of m. It follows that the total number of labels in I_m is $(\log^2 N + \log N)/2$, although the total number of subsets to which m belongs is $O(N)$. For example, in the key tree in Fig. 2(b), node 10 belongs to 16 subsets, i.e., $S_{0,2}$, $S_{0,5}$, $S_{0,6}$, $S_{0,11}$, $S_{0,12}$, $S_{0,13}$, $S_{0,14}$, $S_{0,3}$, $S_{0,7}$, $S_{0,8}$, $S_{0,9}$, $S_{1,3}$, $S_{1,7}$, $S_{1,8}$,$S_{1,9}$, $S_{4,9}$. However, it only needs 6 labels: $\{L_{0,2}, L_{0,3}, L_{0,9}, L_{1,3}, L_{1,9}, L_{4,9}\}$ to calculate $K_{i,j}$ for all of the 16 subsets. In [7], the authors point out that I_m also includes an additional key corresponding to the case

[1] In this paper, a node is a descendant of itself, but not an ancestor of itself.

when all potential members are in the group (i.e., $\mathcal{M} = \mathcal{N}$). As a result, $|I_m| = (\log^2 N + \log N)/2 + 1$.

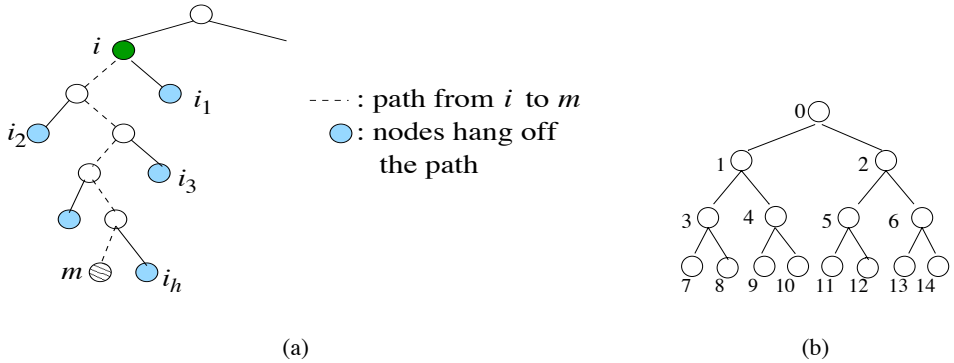


Figure 2: Static SDR key tree

The secure delivery of I_m to m can be done through secure unicast or an off-line method. It is important to note that I_m is the only required information for member m to participate in the group communication for the statelessness of SDR. Also note that I_m is determined by the position of m in the key tree. As a consequence, once a leaf position in the key tree is assigned to a member m, that position cannot be assigned to any other member even when m is currently not in the group. That is the reason why the GCKS in static SDR needs to maintain a key tree large enough for $\mathcal{N}$. Typically, in static SDR, a *returned* member (a member joining the group again after leaving) is assigned to the position that the member was assigned last time.

The number of the resultant subsets of $\mathcal{M}$ is determined by the positions of members of $\mathcal{M}$ in the key tree of size N. Generally speaking, under the assumption that member activity is independent of position in the key tree, the larger the difference between N and M, the more likely that active members are sparsely distributed in the key tree, resulting in $O(M)$ disjoint subsets. On the other hand, the smaller the difference between N and M, the more adjacent the positions of active members in the key tree, resulting in $O(N - M)$ disjoint subsets. As pointed out by [3], given M, the expected number of resultant subsets of $\mathcal{M}$ is $\Theta(\min(N - M, M))$. When $M < N/2$, the expected number of resultant subsets is $\Theta(M)$. Moreover, static SDR requires $O(N \log N)$ storage overhead at the GCKS [3], and $O(\log^2 N)$ storage overhead at the member side.

3 Dynamic SDR

We have seen that static SDR generally requires a very large key tree, which must accommodate all unique members. As a consequence, currently active members, usually a small fraction of all of the unique members, are likely to be widely dispersed in the key tree space. Both of these factors decrease the performance of SDR. In this section, we propose a *dynamic SDR* approach that addresses these two inefficiencies of static SDR.

First, we describe an observation on static SDR, based on which dynamic SDR is proposed.

In the binary key tree of SDR, a node i has a height of h if the subtree rooted at i has 2^h leaves. For example, all leaves have a height of 1. The height of T_i and $S_{i,j}$ are also defined as the height of node i. The size of $S_{i,j}$ is $|S_{i,j}| = |S_i| - |S_j|$.

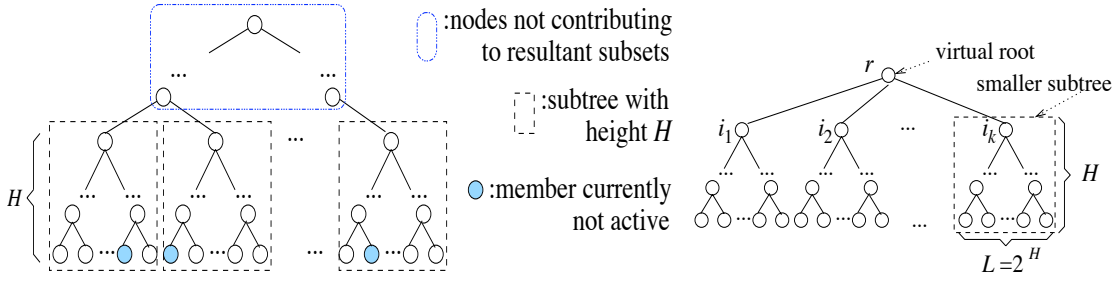
Proposition 1. *In the key tree T of static SDR, if there exists a set of disjoint subtrees $\{T_i\}$ with the same height H such that*

(1) leaves of T=$\bigcup_i$\{leaves of T_i\};

(2) each T_i has at least one leaf in $\mathcal{N}\backslash\mathcal{M}$, i.e., a member currently inactive,

then all resultant subsets of $\mathcal{M}$ have height $h \leq H$ (Fig. 3(a)).

Proof: Since any subtree with a height of H has at least one leaf in $\mathcal{N}\backslash\mathcal{M}$, there is no subtree T_s with a height of $h_1 \geq H$ such that all leaves in T_s are in $\mathcal{M}$. Let $S_{i,j}$ be a resultant subset with height of $h_2 > H$. Let u be the child of i such that j is not a descendant of u. Now u has a height of $h_3 \geq H$. Consider the subtree T_u, all leaves in T_u are now in $S_{i,j}$, and thus in $\mathcal{M}$. We then get a contradiction. ■



(a) Static SDR key tree

(b) Dynamic SDR key tree using subtrees as allocation units

Figure 3: Constructing dynamic SDR key tree

Proposition 1 implies that, if such set of $\{T_i\}$ exists in the key tree of static SDR, all nodes with height

$h > H$ will not contribute to the resultant subsets of $\mathcal{M}$. In other words, the GCKS does not need to maintain nodes with height $h > H$ in the key tree. Instead, the GCKS only needs to maintain a set of smaller subtrees satisfying Proposition 1 with an appropriate height H (we will address how to choose H shortly). The idea of dynamic SDR is to dynamically maintain such set of subtrees.

3.1 Scheme of dynamic SDR

In dynamic SDR, the positions of members are not pre-assigned. Instead, the spaces in the key tree are dynamically allocated and reclaimed, adapting to the current set of active members. More specifically, the GCKS dynamically creates leaves when new member joins, or discards a subtree when all positions of the subtree are inactive. By doing this, the GCKS maintains active members in a dynamic key tree, rather than a large key tree constructed in advance.

For simplicity, we use set of subtrees $\{T_{i_k}\}$ as allocation units, one can view the subtrees connected to a virtual root r (Fig. 3(b)). Initially, the GCKS has a single subtree T_{i_1} connected to the virtual root r. When a member joins the group, regardless of being a new member or a returned member, the member is assigned to the next available position in the key tree (from left to right) and is *unicast* the secret information associated with the new position. The GCKS thereafter encrypts and *multicasts* the updated TEK to the current members in exactly the same way as in static SDR. If new positions are required (step (4) and (5) in Fig. 4), the GCKS creates a new subtree T_{i_k}. When a member, m, leaves the group, the position becomes empty and will never be used by any member (even m itself). And a new TEK is multicast to the members that remain in the group. If all positions of the leftmost subtree become empty, the GCKS discards that subtree. The pseudo code of this process is shown in Fig. 4. Note that step (5) is to maintain the second condition in Proposition 1.

The advantages of maintaining such a dynamic-membership key tree are two-fold. First, instead of maintaining a key tree that is sufficiently large to hold all potential members, dynamic SDR may require a much smaller key tree of a size sufficient to accommodate the maximum number of concurrently active members. This helps reduce key storage cost, both at the members and at the GCKS. Second, dynamic SDR is able to utilize the temporal locality of the members' joining and leaving activity. By assigning members that arrive close in time to positions that are close in the key tree, the GCKS is likely to find a subset that can cover many adjacent members. As a result, a small number of subsets will typically be needed to cover the active members when a new TEK is disseminated. This implies that the messaging overhead associated with rekeying is also reduced. Dynamic SDR achieves the advantages by introducing additional unicast,

```
DynamicSDR(L):
//L = 2^H;
(1)  RightmostSubtree=T_{i_1};
(2)  while (the group is on) {
(3)     case member-join:
(4)        if (no available position in RightmostSubtree)
(5)          or (RightmostSubtree has L − 1 positions attached by active members)
(6)           Create a new subtree T_{i_k};
(7)           RightmostSubtree=T_{i_k};
(8)        Assign the newly joining member to the next available position u in RightmostSubtree;
(9)        Unicast I_u to that member;
(10)       Multicast updated TEK;
(11)    case member-leave:
(12)       Multicast updated TEK;
(13)       if (all L positions of the leftmost subtree are empty)
(14)          Delete the leftmost subtree;
(15) }
```

Figure 4: Key tree constructing algorithm in dynamic SDR

corresponding to deliver I_u to a joining member (step (9) in Fig. 4). However, the overall communication cost (in bytes per second), can be reduced by more than 50% in comparison to that of static SDR, as we will see in Section 4. We will also see the quantitative comparison of the unicast and multicast costs of dynamic SDR in that section.

Since the key tree of dynamic SDR can be extended arbitrarily, dynamic SDR does not require *a priori* knowledge of the size of total member population, N. This avoids the problem, which exists in static SDR, of estimating N. Overestimating N makes the static SDR key tree unnecessarily large, increasing both rekey communication cost and key storage cost, whereas, underestimating N may introduce the problem of having to reject members when all positions have been assigned.

In dynamic SDR, however, the size, $L = 2^H$, of SDR subtree T_{i_k} should be chosen properly, as we describe next.

3.2 Determining L

The choosing of L is a design tradeoff. Based on Proposition 1, one subset covers at most L members. Therefore, when L is small, more resultant subsets are required to cover $\mathcal{M}$. Consequently, the multicast cost increases. When L is large, we may still encounter the space inefficiency of static SDR that active

members disperse in the subtree.

We thus choose L as a reasonable value of $2^{\lceil \log E[M] \rceil}$, where $E[M]$ is the expected value of the number of concurrent members. Ideally, we want to put the concurrent members in one subtree.

3.3 Key storage of dynamic SDR

In dynamic SDR, the size of secret information, $|I_m|$, is reduced from $(\log^2 N + \log N)/2 + 1$ to $(\log^2 L + \log L)/2 + 1$ for the following Proposition.

Proposition 2. *In dynamic SDR, if a member, m, is assigned a position in subtree T_{i_k}, for any resultant subset $S_{i,j}$ covering m, i is a descendant of i_k.*

Proof: Let $S_{i,j}$ be a resultant subset covering m. Node i must be a descendant or an ancestor of i_k. This is true since $m \in S_i$ and m is in T_{i_k}. If i is an ancestor of i_k, i has a height of $h_1 > H$. Then $S_{i,j}$ has a height of $h_2 > H$, which contradicts Proposition 1. Thus i must be a descendant of i_k. ■

A consequence of the Proposition is that, in dynamic SDR, a member only needs to store labels associated with the subtree in which the member is assigned.

Although the key storage size required by a member is fixed when L is chosen, this is not the case for the GCKS. Key storage at the GCKS must be separately analyzed, since it is related to the number of subtrees T_{i_k}. Assuming that the GCKS sequentially assigns the available positions of the key tree from the left to the right to the joining members, we define S as the distance from the leftmost position occupied by an active member to the first available position at the right side. These S positions are referred as the *concurrent spaces*, which determine the key storage at the GCKS. To hold S concurrent spaces, at most $\lceil S/L \rceil + 1$ subtrees are required. Since the GCKS has $2L \log L + 1$ key storage for a subtree T_{i_k} of size L [3], it follows that the key storage at the GCKS is $(\lceil S/L \rceil + 1)(2L \log L + 1)$.

When members arrive according to a Poisson process with rate λ and the time that each active member stays in the group (which is referred as *lifetime*) is exponentially distributed with mean $1/\mu$, the expectation of S can be computed as follows:

$$E[S] = \rho e^{-\rho} \sum_{m=0}^{\infty} \frac{\rho^m}{m!} \sum_{i=1}^{m+1} \frac{1}{i} \tag{1}$$

where $\rho = \lambda/\mu$.

Proof: A concurrent space is allocated when a new member arrives. Thus, the arrival rate of concurrent

spaces is λ, which is the same as the arrival rate of new members. A concurrent space is empty when the corresponding member departs. A concurrent space is removed when it is empty and all "earlier-allocated" concurrent spaces are removed. Let X denote the number of members in the group before a concurrent space is allocated. In $M/M/\infty$, the distribution of X is known as,

$$Pr(X = x) = e^{-\rho}\frac{\rho^x}{x!} \tag{2}$$

Let T denote the sojourn time of a concurrent space. When $X = x$, we have

$$E[T|X = x] = \sum_{i=1}^{x+1}\frac{1}{i\mu} \tag{3}$$

Therefore, we have

$$\begin{aligned} E[T] &= \sum_{x=0}^{\infty} Pr(X = x)E[T|X = x] \\ &= e^{-\rho}\sum_{x=0}^{\infty}\frac{\rho^x}{x!}\sum_{i=1}^{x+1}\frac{1}{i\mu} \end{aligned} \tag{4}$$

Using Little's law, we get

$$E[S] = \lambda E[T] = \rho e^{-\rho}\sum_{x=0}^{\infty}\frac{\rho^x}{x!}\sum_{i=1}^{x+1}\frac{1}{i} \tag{5}$$

■

Using Little's law, the average number of the concurrent members $E[M] = \rho = \lambda/\mu$, thus $E[S]$ can be viewed as a function of $E[M]$, as shown in Fig. 5. The top curve in the figure presents $E[S]$ as a function of $E[M]$ according to (1), which shows that $E[S]$ increases super-linearly with $E[M]$.

$E[S]$ also depends on the distribution of members' lifetime. If members' lifetime is deterministic (e.g., members are First-In-First-Out), all currently active members will occupy consecutive positions in the key tree. In this case, $E[S]$ is identical to $E[M]$. Generally, the higher variance members' lifetime has, the larger $E[S]$ is.

A large value S results in an increased key storage at the GCKS side. Also, currently active members disperse in the key tree as S increases, incurring more resultant subsets and thus more rekeying messages. As a result, it is desirable to keep S small, a topic we address next.

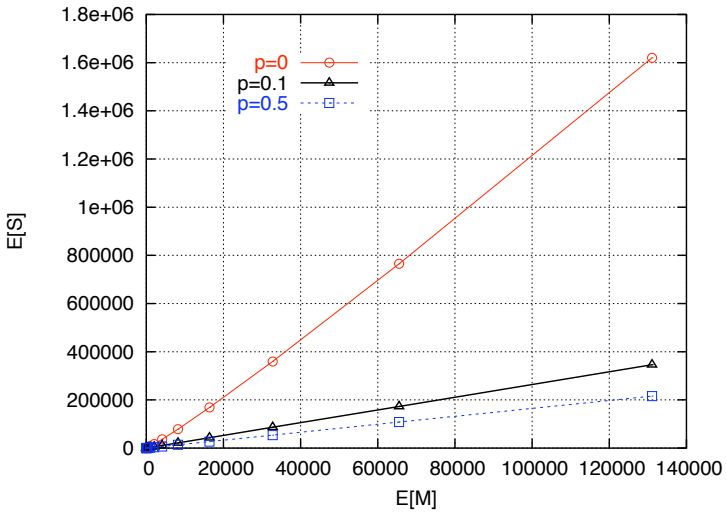


Figure 5: Relationship between $E[S]$ and $E[M]$

3.4 Reducing S by shifting

In this subsection, we propose a simple operation, namely *shifting*, that can be used to reduce S.

We define *shifting* as the operation of detaching the leftmost active member in the key tree and re-attaching the member to the next available position (for new arrivals) in the key tree. This is based on the consideration that in a dynamic SDR subtree, a leaf position assigned to a member cannot be assigned again. When some holes (i.e., positions with departed members) are generated in the key tree, shifting the leftmost active member may reduce the concurrent spaces, S, and make active members more adjacent, as illustrated in Fig. 6.

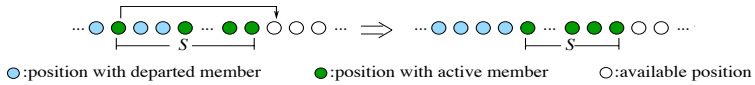


Figure 6: Shifting member from left to right reduces S by 2

When active members are shifted, they are delivered new secret information associated with the new positions by unicast. From the collusion-proof property of static SDR, we can show that shifting does not jeopardize the confidentiality of the group communication – we will discuss this in detail in Section 5.

There are many strategies for shifting. Here, we investigate two approaches, namely *probabilistic shifting* and *threshold-based shifting*. We start with probabilistic shifting, since it is simpler and also easier to analyze.

3.4.1 Probabilistic shifting

Probabilistic shifting is defined as follows. When a member joins the group, that member is assigned to the next available position in the key tree. Meanwhile, with a probability p, the GCKS shifts the leftmost member, i.e., the leftmost active member is detached and re-attached to the position just to the right of the newly arrived member. Note that the shifting probability p affects the tradeoff between the shifting cost, associated with unicasting the secret information for the new position to the shifted member, and the value of S. When $p = 0$, probabilistic shifting is identical to dynamic SDR without shifting, where the shifting cost is zero while S is high. When $p = 1$, whenever a new member joins the group, the leftmost member is shifted. In this case, S can be small while the shifting cost is relatively high. In practice, the shifting probability p is a parameter set by the GCKS and can be application specific.

When members join the group according to a Poisson process with rate λ and a member's lifetime is exponentially distributed with mean $1/\mu$, we can compute the expected number of the concurrent spaces, $E(S)$, as a function of the shifting probability p:

$$E[S] = \rho e^{-\rho} \sum_{m=0}^{\infty} \frac{\rho^m}{m!} \left(\sum_{i=1}^{m+1} \frac{1}{i + p\rho} + p \sum_{i=1}^{m} \frac{1}{i + p\rho} \right) \tag{6}$$

where, $\rho = \lambda/\mu$.

Proof: To make calculation easier, we adopt a slightly different probabilistic shifting algorithm without changing the value of S. The difference is that, when X is 0 upon the arrival of a new member, the new member would be treated as existing members and would be shifted with probability p at the same time.

A concurrent space is allocated when a new member arrives or a existing member shifts. Thus, the arrival rate of concurrent spaces is $\lambda(1 + p)$, which is the sum of the arrival rate of new members and the shifting rate of existing members.

Let p_N denote the probability that a concurrent space is occupied by a new member, and let p_S denote the probability that a concurrent space is occupied by a shifted member. We have

$$p_N = \frac{1}{1+p} \quad , \quad p_S = \frac{p}{1+p} \tag{7}$$

Let T_N denote the sojourn time of a concurrent space occupied by new members, and let T_S denote the

sojourn time of a concurrent space occupied by shifted members. We have

$$E[T_N|X=x] = p\sum_{i=1}^{x}\frac{1}{i\mu+p\lambda} + (1-p)\sum_{i=1}^{x+1}\frac{1}{i\mu+p\lambda} \quad (8)$$

$$E[T_S|X=x] = \sum_{i=1}^{x+1}\frac{1}{i\mu+p\lambda} \quad (9)$$

$$E[T_N] = \sum_{x=0}^{\infty} Pr(X=x)E[T_N|X=x] \quad (10)$$

$$E[T_S] = \sum_{x=0}^{\infty} Pr(X=x)E[T_N|X=x] \quad (11)$$

Therefore, we get

$$\begin{aligned} E[T] &= p_N E[T_N] + p_S E[T_S] \\ &= \frac{e^{-\rho}}{1+p}\sum_{x=0}^{\infty}\frac{\rho^x}{x!}(\sum_{i=1}^{x+1}\frac{1}{i\mu+p\lambda} + p\sum_{i=1}^{x}\frac{1}{i\mu+p\lambda}) \end{aligned} \quad (12)$$

Using Little's law, we get

$$E[S] = \lambda E[T] = \rho e^{-\rho}\sum_{x=0}^{\infty}\frac{\rho^x}{x!}(\sum_{i=1}^{x+1}\frac{1}{i+p\rho} + p\sum_{i=1}^{x}\frac{1}{i+p\rho}) \quad (13)$$

■

Fig. 5 presents the result of $E[S]$ for $p = 0, 0.1$ and 0.5 respectively. We observe that $E[S]$ decreases as a function of p. In particular, when the expected group size $E[M] = 100K$, the space-member-ratio, $E[S]/E[M]$ is 12.11 for dynamic SDR without shifting, and reduces to 2.64 when the shifting probability is 0.1, and further reduces to 1.65 when the shifting probability is 0.5. Thus, by introducing a small shifting probability p, dynamic SDR can effectively reduce the average concurrent space S, therefore reducing the key storage cost at the GCKS and potentially the rekey cost as well.

We have also investigated probabilistic shifting upon members' departure and shifting according to a fixed-rate Poisson process independent with members' arrival or departure. The results (not shown due to space constraint) are qualitatively similar to that of the probabilistic shifting with respect to members' arrival. We next consider a different kind of shifting strategy – threshold-based shifting.

3.4.2 Threshold-based shifting

We define the occupancy ratio γ as the number of active group members to the number of concurrent spaces, i.e., $\gamma = M/S$. Informally, the occupancy ratio measures the compactness of the active group members in the key tree space. The larger the occupancy ratio is, the more likely the members are adjacent to each other in the key tree, and thus can be covered by fewer subsets. To keep the rekey process efficient, the GCKS should keep the occupancy ratio high. A natural way to achieve this is to define a threshold $\Gamma < 1$; when a member leaves the group, the GCKS computes the occupancy ratio γ and compares it to the threshold Γ. If the occupancy ratio falls below the threshold (i.e., $\gamma < \Gamma$), the GCKS will keep shifting the leftmost member until $\gamma \geq \Gamma$. We refer to this strategy as threshold-based shifting.

As with the shifting probability p in the probabilistic shifting scheme, the threshold Γ is also an application-specific parameter, whose value affects the tradeoff between the shifting cost (unicast) and the rekey cost (multicast). Analyzing the tradeoff is difficult and may vary for different application scenarios. Thus, we evaluate the choice of different thresholds through simulation, as discussed in Section 4.2.

3.5 Block alignment

So far, we have treated the newly arrived members and the shifted members identically when assigning a member to an available position. However, a member that has been in the group for a long time and has become the leftmost member in the key tree may have very different characteristics in terms of the remaining service time (the duration that a member remains in the group), than that of a member who just joined the group. For example, if the distribution of lifetime has a *decreasing failure rate*, the longer a member stays in the group, the more likely this member will continue to stay in the group. Thus, it is preferable to separate the shifted members and the newly arrived members so that members with similar remaining service time can be assigned to positions adjacent to each other.

Given the above considerations, we further propose an enhancement to the dynamic SDR with shifting scheme by allocating different blocks in the key tree for new members and shifted members. Fig. 7 presents the pseudo-code for determining member's position in the key tree. In the enhanced scheme, the positions in the key tree are divided into blocks, each of which has size $B = 2^k$ and boundaries at positions that are multiples of B. The positions within each block are either for shifted members only or for newly joined members only. Furthermore, the GCKS always attaches a member to the next available position within the block that corresponds to the status of the member (shifted or newly arrived). When all positions in one

block are occupied, the GCKS will assign the member the first position in the next available block.

B is an application-specific parameter similar to p and Γ. We evaluate the effects of such block alignment through simulation. The result will be presented in Section 4.2.

```
GetPosition(flag):
(1)  static pos_new = 1;
(2)  static pos_shift = B + 1;
(3)  static pos_avail = 2B + 1;
(4)  switch (flag) {
(5)      case join:
(6)          retval = pos_new;
(7)          pos_new ++;
(8)          if (pos_new %B == 1) //block is full
(9)              pos_new = pos_avail;
(10)             pos_avial += B;
(11)     case shift:
(12)         retval = pos_shift;
(13)         pos_shift ++;
(14)         if (pos_shift %B == 1) //block is full
(15)             pos_new = pos_avail;
(16)             pos_avial += B;
(17)   }
(18) return retval;
```

Figure 7: Computing member's position with block size B

4 Evaluation

In this section, we evaluate the performance of the dynamic SDR through simulation. We will first describe the simulation model and the performance metrics, then present the results.

4.1 Simulation model and performance metrics

We assume a fixed total population $N = 2^{17}$ to which a GCKS provides key management, i.e., the number of potential members is 2^{17}. Each member independently decides to join or leave the group. We approximate the members' arrival by a fixed-rate Poisson process and assume that the lifetimes are independently and identically distributed random variables. In the simulation, we evaluate different lifetime distribution

functions, which include exponential, lognormal and uniform distribution.

In the following, we will present results for immediate-rekey scheme, i.e., the GCKS conducts group rekey immediately after a member joins/leaves the group. The result for batch-rekey scheme is similar (not presented due to space limit), since the performance of SDR is insensitive to the rekey period [3].

The performance metrics of interest are the *key storage cost* (both at the member side and at the GCKS side) and *rekey communication cost*. We repeat each simulation with different seed five times and take the average. The key storage cost is measured as the key storage described in the previous section. More specifically, we assume to use 3DES for encryption and each label has 128-bits of storage.

The rekey communication cost is measured as the number of unit-size rekey messages (assuming one message contains one 128-bit key) per unit time, which is further divided into *multicast cost* and *unicast cost*.

The multicast cost includes the messages containing the updates for TEK that are multicast to the active members. Since the GCKS needs to send one encrypted TEK for each subset, the multicast cost for distributing one update of TEK can be computed by the minimum number of subsets used to cover the active members in the key tree. Since multicast rekeying occurs when a member joins or leaves the group, and when the system is in steady state, the rate at which members depart the group should equal to the rate that members join the group, we can compute the overall multicast cost C_M as follows:

$$C_M = 2\lambda\overline{N_{SD}}$$

where λ is the arrival rate and $\overline{N_{SD}}$ is the average number of subsets that the GCKS uses for one TEK update.

The unicast cost includes the messages for delivering the secret information to a joining member or a shifted member. Thus, the unicast cost can be computed by the key storage at the member side.

For static SDR scheme, the secret information is delivered to a user when that user joins the group for the first time. Since a member's position in the key tree is fixed, no additional unicast costs are incurred when the member returns to the group. To favor static SDR, we assume that the system has been running for long enough so that each member has received the secret information for its position. Thus, we count the unicast cost for static SDR as zero.

For dynamic SDR schemes, the unicast cost is computed as follows.

$$C_U = (\lambda + v)N_K$$

where v is the rate that members are shifted and N_K is the key storage at the member side and equals to $(\lceil \log E[M] \rceil^2 + \lceil \log E[M] \rceil)/2 + 1$ based on the analysis result in Section 3.

The overall rekey communication cost of dynamic SDR is the weighted sum of the multicast cost and the unicast cost. In the rest of the evaluation, we treat the cost of unicasting a message the same as that of multicasting, even though the unicast cost should be much lower than the multicast cost with respect to the number of links that the message travels. Some more discussion on the relative weight of the unicast cost and multicast cost is included in Section 4.2.4.

In summary, the overall rekey communication cost of static SDR is

$$C_s = 2\lambda \overline{N_{SD}^s} \tag{14}$$

where $\overline{N_{SD}^s}$ is the average number of subsets using static SDR for one TEK update.

The overall rekey communication cost of dynamic SDR is

$$C_d = 2\lambda \overline{N_{SD}^d} + (\lambda + v)N_K \tag{15}$$

where $\overline{N_{SD}^d}$ is the average number of subsets using dynamic SDR for one TEK update.

4.2 Simulation results

4.2.1 Effects of group size

We have conjectured that dynamic SDR would have advantages over static SDR when the group size (the number of concurrently active members) is much smaller than the total population (the number of distinct potential members). Now we evaluate this conjecture by comparing the performance of static SDR and dynamic SDR for different group sizes.

With a fixed total population ($N = 2^{17} = 131072$) and a fixed mean lifetime ($1/\mu = 100$), we use different value of the arrival rate (λ) to vary the expected group size ($E[M] = \lambda/\mu$) from 100 to 1.3×10^5. For now, we only consider the dynamic SDR scheme without shifting.

Fig. 8 compares the key storage of static SDR and dynamic SDR for different group size with exponentially-distributed lifetimes. The top two curves represent the key storage cost at GCKS and the bottom two curves represent the key storage cost at member side. Since static SDR maintains a fixed key tree whose size is determined by the total member population, its key storage costs, at both GCKS and member side, are invariant with different group size. For dynamic SDR, however, the key storage increases when the expected group size increases. Compared to static SDR, the member-side key storage cost of dynamic SDR is consistently lower, since M is always smaller than N. However the GCKS key storage cost of dynamic SDR begins to exceed that of static SDR as $E[M]$ increases. This is because the expected size of concurrent spaces $E[S]$, which includes both active members in the key tree and departed members in between, becomes larger than N when $E[M]$ is significant ($> 10\%$) compared to N.

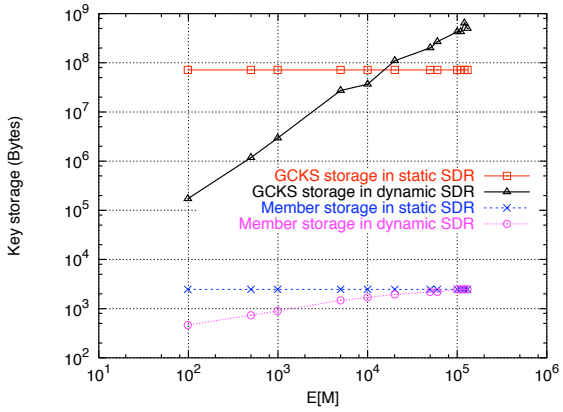


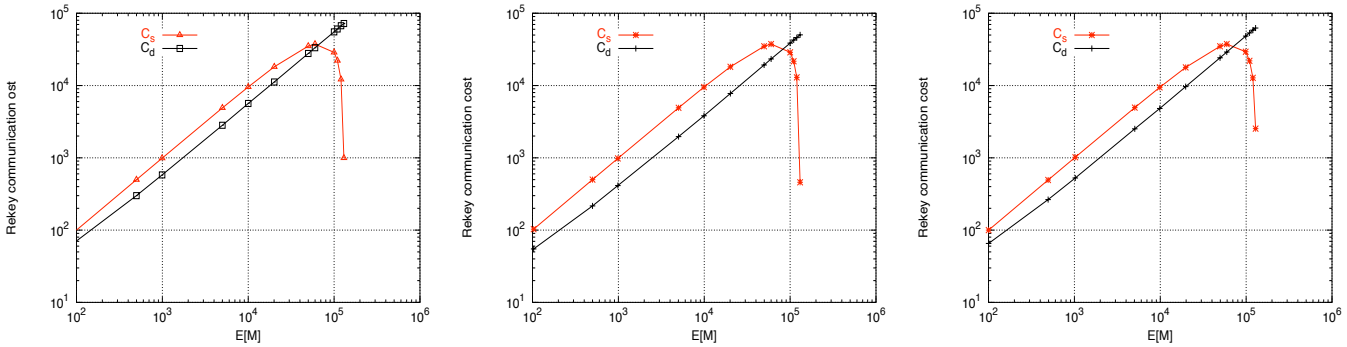
Figure 8: Key storage of static SDR and dynamic SDR for different group size

We next compare the rekey communication cost of static SDR, C_s, and dynamic SDR, C_d, as described in (14) and (15) respectively. Fig. 9(a), 9(b) and 9(c) present the results for scenarios where the lifetime distribution functions are exponential, uniform and lognormal, respectively. First, one can see that the results for three scenarios are qualitatively similar. We also observe that the rekey communication cost of static SDR increases as $E[M]$ increase, reaching a maximum when $E[M]$ is about $N/2$, and then starts to decrease when $E[M]$ gets close to N. This behavior matches well the reasoning in Section 2 and is consistent with the result in [3]. Furthermore, we observe that when $E[M] < 2 \times 10^4$, the rekey communication cost C_d is much lower, nearly half, compared to C_s. This demonstrates the benefit of utilizing the temporal locality of members' activities, as in dynamic SDR. Only when $E[M]$ is greater than $N/2$, does C_s outperform C_d. Again, this is because that members in static SDR are spatially constrained in the key tree and are naturally adjacent to each other, resulting a smaller rekeying cost, while the distribution of members in dynamic SDR

Table 1: Parameters of the simulation configurations

Config. ID	Poisson arrival	Distr. of lifetime	Mean of lifetime	Var. of lifetime
EXP	$\lambda = 10$	exponential	100	10^4
UNI	$\lambda = 10$	uniform	100	3.3×10^3
LOG_l	$\lambda = 10$	lognormal	100	10^3
LOG_m	$\lambda = 10$	lognormal	100	10^4
LOG_h	$\lambda = 10$	lognormal	100	10^5

key tree does not change when $E[M]$ is close to N.



(a) Probabilistic shifting scheme (b) Threshold-based shifting scheme (c) Threshold-based shifting scheme

Figure 9: Rekey communication cost of static SDR and dynamic SDR for different group size

We have seen the benefit of dynamic SDR when $M \ll N$. In fact, many practical applications have this property. For example, the MBone STS-71 session has $M \approx 360$ while having $N \approx 4000$ [1]. Another example is in pay-per-view service: the number of people watching a movie at the same time, M, is usually orders of magnitude smaller than the total number of people having cable TV, N. In such cases, dynamic SDR will be useful.

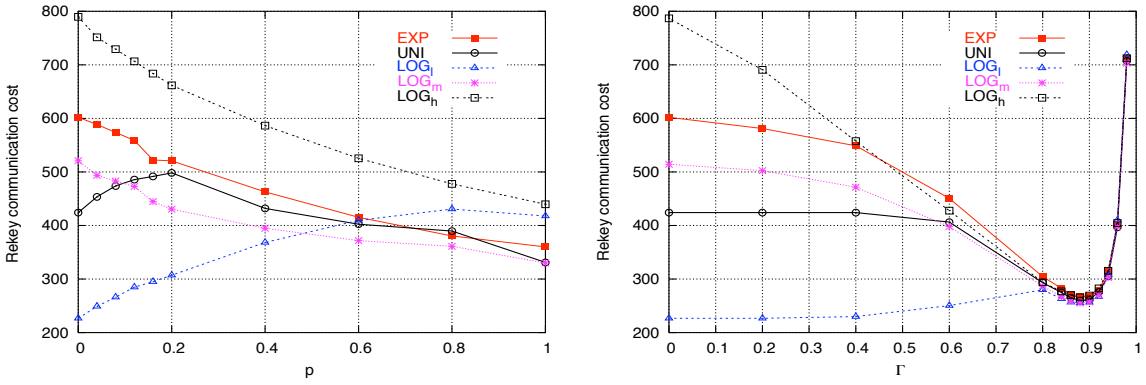
In the next, we will focus on the scenario where $M \ll N$ and evaluate the impact of shifting and block alignment.

4.2.2 Impact of shifting

In this subsection, we study the performance of dynamic SDR with shifting. We consider the case where $E[M] = 1000 (\ll N)$. Table 1 shows the five different configurations that we use to obtain the simulation results[2]. We simulate probabilistic shifting and threshold-based shifting strategies for each configuration.

[2]For other λ and μ that achieve the same $E[M]$, we have the similar results.

Figure 10(a) shows the communication cost, C_d, of dynamic SDR with probabilistic shifting for the five different configurations. We observe that, without shifting ($p = 0$), the rekey communication cost tends to be higher when members lifetime is of high variance.



(a) Probabilistic shifting scheme

(b) Threshold-based shifting scheme

Figure 10: Rekey communication cost of dynamic SDR combined shifting on configurations with $E[M] = 1000$

With shifting ($p > 0$), we find two different kinds of behaviors among the five configurations. For configurations EXP, LOG_m and LOG_h, which have high variance lifetime, increasing the shifting probability generally reduces the rekey communication cost C_d. This is because, with no shifting, S is large; while introducing shifting, although unicast cost is increased, S can be substantially reduced, which results in a reduced multicast cost. Since multicast cost (> 200) dominates unicast cost (ranging from 28 to 56), the overall rekey communication cost is also reduced.

For low variance configurations, UNI and LOG_l, when increasing the shifting rate, the communication cost C_d does not necessarily decrease. In these cases, S is close to M without shifting. Shifting cannot reduce S much. On the contrary, shifting might affect the distribution of members in the key tree, which may increase the number of subsets needed to cover the active members. In an extreme example where member's lifetime is deterministic (variance is zero), without shifting, active members are always adjacent to each other in the key tree; adding shifting may break this pattern and may require more subsets to cover the active members. As a result, the choice of optimal value of p depends on the lifetime distribution.

Fig. 10(b) shows the evaluation of threshold-based shifting scheme applied to the five configurations in Table 1. In the figure, it appears that, all configurations have a local minimum rekey communication cost when $\Gamma \approx 0.9$. When Γ goes beyond 0.9, there is a dramatically increase of the rekey cost. This is

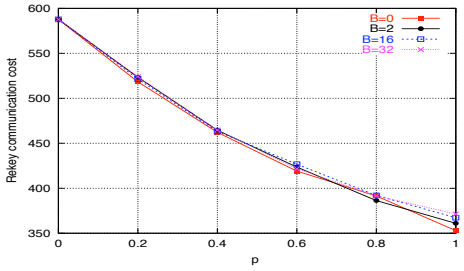
because, for a threshold $\Gamma > 0.9$, even though multicast cost may be reduced, unicast cost associated with frequent shifting becomes so high that the overall rekey cost is dramatically increased. Except for LOG_t, the local minimum rekey cost is also the global minimum. For configuration LOG_t, although the global minimum cost is achieved when $\Gamma = 0$, the local minimum rekey cost at $\Gamma \approx 0.9$ is very close to the global minimum. Thus choosing a proper value of the threshold parameter Γ is not as sensitive to members' lifetime distribution as in probabilistic shifting.

4.2.3 Enhancement with block alignment

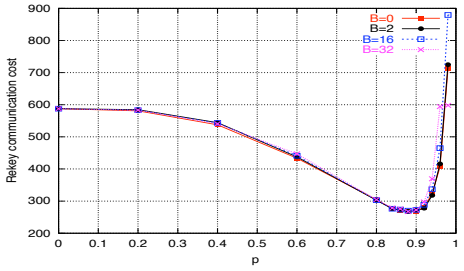
We next evaluate the performance of block alignment as an enhancement to probabilistic shifting and threshold-based shifting for the five simulation configurations. Here we present the results of configuration EXP and LOG_h.

Fig. 11 plots the rekey communication cost when applying block alignment with $B = 0, 2, 16$ and 32 for probabilistic shifting and threshold-based shifting. We observe that, for configuration EXP, introducing block alignment (with various B) does not have much improvement on reducing rekey communication cost (sometimes is even worse). This is due to the memoryless property of exponential distribution – a new member has the same distribution of the remaining service time as shifted members. Thus assigning new members and shifted members to positions in different blocks is of no help. However, for configuration LOG_h, the improvement of block alignment is significant. For example, using probabilistic shifting with $p = 1$, rekey communication cost is 440 without alignment ($B = 0$), and reduces to 390 when $B = 2$, and further reduces to 350 when $B = 16$. Furthermore, for this particular group size, we find that increasing block size B beyond 16 does not provide much additional improvement. In practice, it is sufficient to use a small value, 16 or 32, for B, so as to achieve most performance gain.

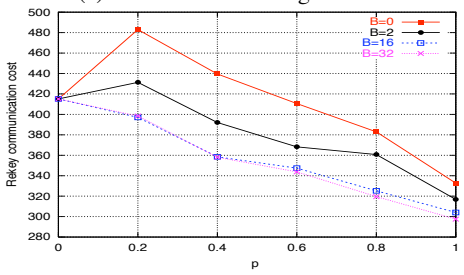
Compared to threshold-based shifting, improvement of using block alignment is more evident in probabilistic shifting. This is because that, in probabilistic shifting, only one member is shifted at a time; thus it is more likely that new members and shifted members interleave in the key tree. Block alignment helps in avoiding this situation. But for threshold-based shifting, many members may be shifted at a time, making them naturally adjacent to each other after shifting. Thus the effect of block alignment is limited.



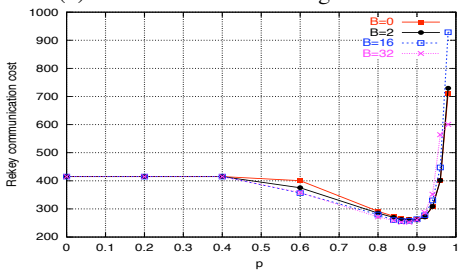
(a) Probabilistic shifting on EXP



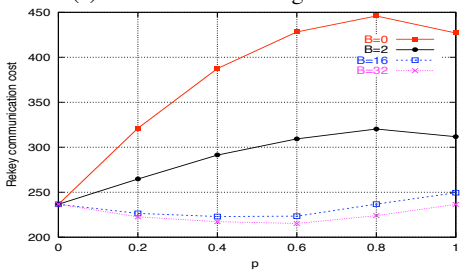
(b) Threshold-based shifting on EXP



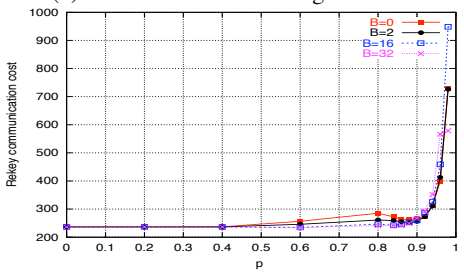
(c) Probabilistic shifting on $UNIF$



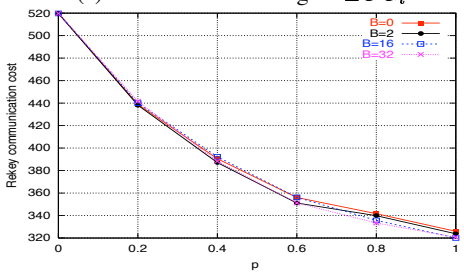
(d) Threshold-based shifting on $UNIF$



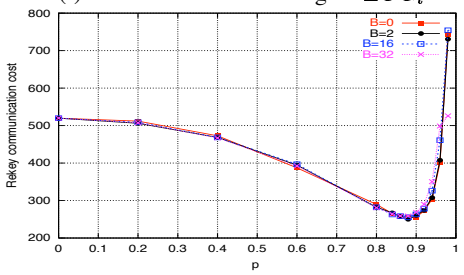
(e) Probabilistic shifting on LOG_l



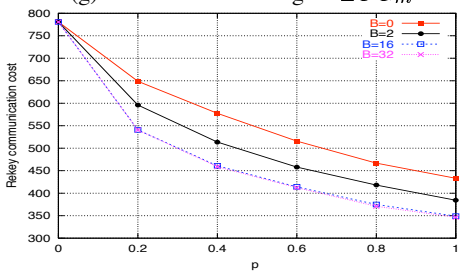
(f) Threshold-based shifting on LOG_l



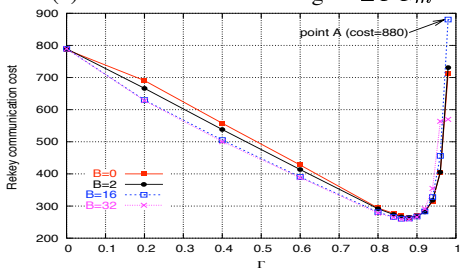
(g) Probabilistic shifting on LOG_m



(h) Threshold-based shifting on LOG_m

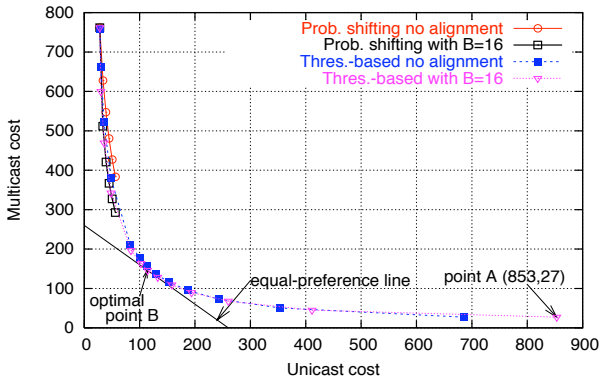


(i) Probabilistic shifting on LOG_h

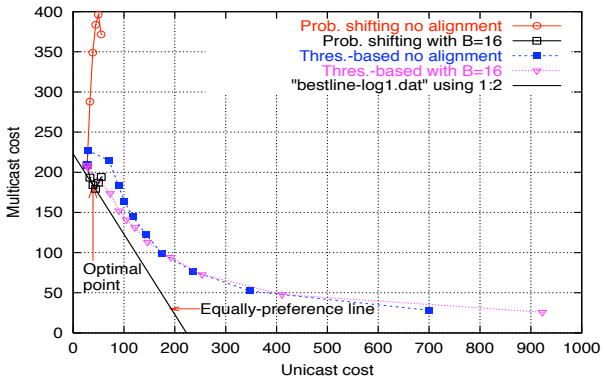


(j) Threshold-based shifting on LOG_h

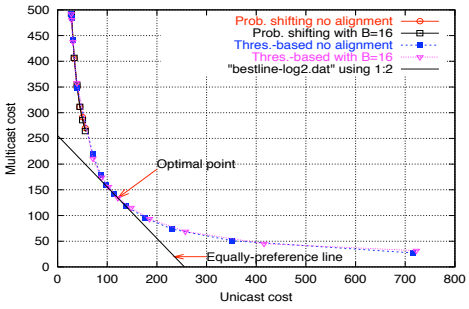
Figure 11: Cost of dynamic SDR using shifting and alignment on the five configurations with $E[M] = 1000$



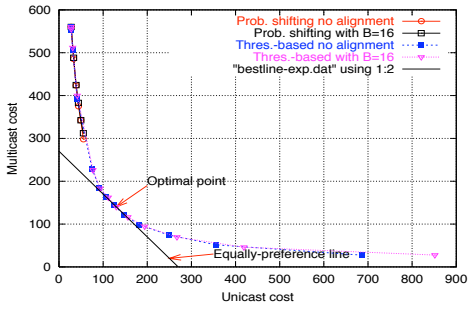
(a) LOG_h



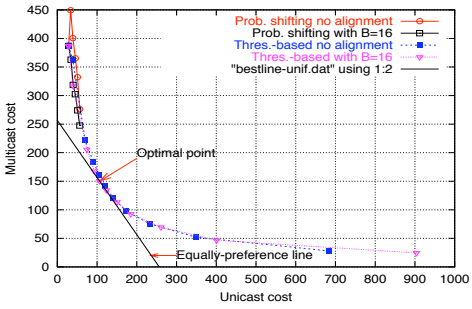
(b) LOG_l



(c) LOG_m



(d) EXP



(e) $UNIF$

Figure 12: Tradeoff between unicast cost and multicast cost for different shifting schemes with $E[M] = 1000$

4.2.4 Tradeoff between unicast cost and multicast cost

As described in Section 3, dynamic SDR reduces multicast costs by introducing additional unicast, by which the secret information is delivered to shifted or returned members. Since the different strategies proposed in this paper have different parameters (p or Γ) to configure, each of which reflects the tradeoff between unicast cost and multicast cost. To compare different schemes, we study the tradeoff graph of these proposed schemes as shown in Fig. 12. In the tradeoff graph, a point on a curve denotes the multicast and unicast cost for the corresponding strategy with a particular parameter. For example, point A in Fig. 12 is associated with unicast cost of 853 and multicast cost of 27, denoting the total cost of 880 for the threshold-based shifting combined alignment of B=16 with parameter $\Gamma = 0.98$ (point A in Fig. 11(j)).

From the figure, we observe that reducing multicast cost comes at a cost of increasing unicast cost, and vice versa. The relative weight of unicast cost and multicast cost affects the choice of the optimal schemes

and the operating parameters. If we treat unicast cost as expensive as multicast cost, in the tradeoff graph, all points on a line with a slope -1 are equally preferable. While points on a line close to point (0,0) are preferred over points on lines far away. In this sense, the threshold-based shifting combined alignment with block size B=16 offers the best tradeoff among the algorithms considered, achieving an optimal value with multicast cost of 146 and unicast cost of 114 (point B in Fig. 12(a)). In general, if the relative weight of unicast cost and multicast cost is w, the equal-preference lines will have slope $-w$ in the tradeoff graph. In this case, the best approach and the optimal parameters may be different.

Fig. 12(b), Fig. 12(c), Fig. 12(d) and Fig. 12(e) present tradeoff on configurations LOG_l, LOG_m, EXP and $UNIF$ respectively. Although the tradeoff graphs of the five configurations in Fig. 12 are slightly different for probabilistic shifting, they have the similar tradeoff for threshold-based shifting. Except for configuration LOG_l, threshold-based shifting works better than probabilistic shifting and the corresponding optimal points are associated with threshold-based shifting combined alignement with block size $B = 16$. For configuration LOG_l, the optimal point is associated with probabilistic shifting combined alignment with block size $B = 16$.

5 Discussion

In Section 4, we have focused on the performance aspect of dynamic SDR. In this section, we examine other important properties of dynamic SDR, in particular, security and multicast stateless property of dynamic SDR.

5.1 Security

In this paper, we use three criteria to measure security: forward confidentiality, backward confidentiality and collusion problem. As we know, static SDR maintains forward/backward confidentiality, and has no collusion problem. We find that those properties hold for dynamic SDR.

Below is the brief of our proof. One key difference between dynamic SDR and static SDR is that a member might have the secret information for multiple positions in dynamic SDR. We project dynamic SDR onto static SDR by mapping the member ID space from one dimension into two dimensions, i.e., member m at ith position in the dynamic SDR is mapped to member (m, i) in static SDR. Thus, member m in the dynamic SDR can be viewed as coalition of members (m, i) for all i in static SDR. Furthermore, coalition among member m_1,...,m_j in the dynamic SDR can be viewed as coalition of members

$(m_1, 1)$,...,(m_1, i_1),...,...,$(m_j, 1)$,...,(m_j, i_j) in static SDR. Since in static SDR, forward/backward confidentiality, and collusion-proof are ensured, i.e., any coalition of members cannot acquire any unauthorized information, we conclude that in the dynamic SDR, forward/backward confidentiality, and collusion-proof property are preserved.

5.2 Multicast Stateless

Among all group key management algorithms, static SDR belongs to the so-called stateless algorithms. In static SDR, rekeying messages only contain updated TEKs encrypted with unchanged KEKs. As a result, members do not need to keep track of history of rekeying. This is not true for stateful key management algorithms such as LKH, where rekeying messages may contain updated KEKs encrypted using previous KEKs.

In dynamic SDR, KEKs (keys of resultant subsets) are long-lived and can be computed based on the secret information securely unicast to members. Multicast messages in dynamic SDR distribute only TEKs. The multicast messages are not required to be reliably delivered to members so as to maintain their states correctly. However, in dynamic SDR, the GCKS is required to reliably unicast a member the secret information for the new position when the member joins or shifts. In other words, only unicast transport of state is sufficient. For this reason, we classify dynamic SDR as a multicast stateless algorithm. The multicast stateless property is useful in practice since providing reliable multicast is still a challenging problem.

Note that static SDR keeps the complete statelessness at the cost of performance due to space inefficiency.

6 Related Work

Most scalable centralized key-management algorithms make use of a tree structure to manage members. These algorithms could be broadly divided into *stateful* algorithms and *stateless* algorithms depending on whether members need to track the communication history to participate in the group communication.

In stateful algorithms ([2, 5, 9, 10]), the active members are leaves of the tree. While in stateless algorithms ([7]), the potential members are leaves of the tree.

LKH is a stateful algorithm. In an LKH tree, there is a leaf node corresponding to each active member. There is a key associated with each node in the tree, and each member holds a copy of every key on the path

from its corresponding leaf node to the root of the tree. Hence, the key corresponding to the root node is shared by all members, and serves as the TEK.

Static SDR is a stateless algorithm. In a static SDR tree, there is a leaf node corresponding to each potential member. Subsets are defined through the tree, and each member holds subset keys for all subsets to which it belongs.

The performance of key-management algorithms is mostly determined by the positions of concurrent members in the tree. [4, 11] propose methods to improve the performance of LKH by adjusting the positions of members dynamically so as to balance the key tree and reduce the overall height. Our work aims to improve the performance of SDR using the similar methodology – dynamically adjusting the positions of members.

A performance comparison between static SDR and LKH is given in [3]. Both the key storage and the rekey communication cost are compared in different scenarios, e.g. immediate rekeying, periodical batch rekeying and membership batch rekeying. It was shown that static SDR outperforms LKH in batch rekeying while LKH outperforms static SDR in immediate rekeying.

7 Conclusion

Static Subset Difference Revocation (SDR) is the current state of the art in stateless group rekeying algorithms. However, it works inefficiently when the number of the active members in the group is much less than the number of potential members, which is the case in many practical applications.

In this paper, we have proposed a group rekeying algorithm, dynamic SDR, which still keeps multicast stateless without the requirement of estimating the number of all potential members. By dynamically constructing the key tree, dynamic SDR uses a smaller key tree sufficiently large for the currently active members rather than the potential members. The smaller key tree reduces both the key storage cost and rekey communication cost compared to static SDR. We also introduce some enhancements to further improve the performance of dynamic SDR. Our evaluation shows that dynamic SDR significantly improves the performance of static SDR, reducing by half the rekey communication cost in the case that the number of the currently active members is much less than the total number of potential members. Also, compared to static SDR, dynamic SDR does not need to know the maximum number of potential group members in advance, a value that can be difficult to estimate in practice.

References

[1] K. C. Almeroth and M. H. Ammar. Collecting and modeling the join/leave behavior of multicast group members in the MBone. In *HPDC*, pages 209–216, 1996.

[2] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key management for secure internet multicast using boolean function minimization techniques. In *Proceedings IEEE Infocomm'99*, volume 2, pages 689–698, 1999.

[3] W. Chen and L. R. Dondeti. Performance comparison of stateful and stateless group rekeying algorithms. In *Fourth International Workshop on Networked Group Communication (NGC'02)*, Boston, MA, October, 2002.

[4] L. 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] 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, 1998.

[6] S. Mittra. Iolus: A framework for scalable secure multicasting. In *SIGCOMM*, pages 277–288, Cannes, France, 1997.

[7] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. *Lecture Notes in Computer Science*, 2139:41–62, 2001.

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

[9] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. IETF, RFC 2627,
`http://www.faqs.org/rfcs/rfc2627.html`.

[10] C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. In *ACM SIGCOMM'98*, pages 68–79, Vancouver, Canada, September, 1998.

[11] Y. Yang, X. Steve Li, X. Zhang, and S. S. Lam. Reliable group rekeying: A performance analysis. In *ACM SIGCOMM'01*, pages 27–38, San Diego, CA, 2001.