# Optimal Subset-Difference Broadcast Encryption with Free Riders

Murat Ak, Kamer Kaya, Ali Aydın Selçuk

Department of Computer Engineering
Bilkent University
Ankara, 06800, Turkey
{muratak,kamer,selcuk}@cs.bilkent.edu.tr

September 10, 2008

**Abstract**

Broadcast encryption (BE) deals with secure transmission of a message to a group of receivers such that only an authorized subset of receivers can decrypt the message. The transmission cost of a BE system can be reduced considerably if a limited number of free riders can be tolerated in the system. In this paper, we study the problem of how to optimally place a given number of free riders in a subset difference (SD) based BE system, which is currently the most efficient BE scheme in use and has also been incorporated in standards, and we propose a polynomial-time optimal placement algorithm and three more efficient heuristics for this problem. Simulation experiments show that SD-based BE schemes can benefit significantly from the proposed algorithms.

**Keywords:**Broadcast encryption; subset difference scheme; free riders

## 1 Introduction

Today's secure multimedia applications such as pay-TV, content protection, secure audio streaming and Internet multicasting usually require a broadcast encryption (BE) scheme which enables transmitting data to a large set of receivers such that only an authorized subset can decrypt it. This is typically achieved by pre-establishing a set of long-term keys at each receiver device, which is later used to support or revoke selected sets. The particular design of a BE system varies according to the system characteristics, such as size of the user domain, required security level, available bandwidth, and hardware capabilities. In the traditional setting, the amount of long-term storage is very limited as it has to be tamper resistant, the communication channel is one way, and the devices are stateless in the sense that no additional long-term storage is possible.

Although recent advances in the technology such as availability of two-way communication channels have reduced the pay-per-view TV systems' reliance on BE schemes, new application areas emerged that benefit from BE greatly such as content protection [9, 12], multicasting promotional material and low cost pay-per-view events [2], multi-certificate revocation/validation [3] and dynamic group key management [13, 14].

Two important performance parameters in evaluating a BE system is the key storage and transmission overheads incurred. Some of the most efficient BE schemes today are the subset difference (SD) scheme of Naor et al. [10] and its variants [6, 7]. The SD scheme has become popular in applications recently and is already implemented in the next-generation DVD standard [1].

In the traditional BE model, it is assumed that all unauthorized receivers must be excluded in a broadcast. Abdalla et al. [2] observed that this model is unnecessarily strict for most practical applications and the cost of a BE system can be reduced significantly when some free riders can be tolerated.

## 1.1 Related Work

After Berkovits [4] introduced the idea of BE in 1991, Fiat and Naor [5] presented their model which is the first formal work in the area. They introduced the *resiliency* concept, and defined *k-resilience* to mean being resilient against a coalition of up to $k$ revoked users. Their best scheme required every receiver to store $O(k \log k \log n)$ keys and the center to broadcast $O(k^2 \log^2 k \log n)$ messages where $n$ is the total number of users.

Wallner et al. [13] and Wong et al. [14] independently proposed the logical key hierarchy (LKH) for secure Internet multicast. LKH was not a broadcast encryption scheme, but its key distribution idea was very useful for broadcast encryption. The idea was to relate the receivers with the leaves of a tree, associate a unique key with each node of the tree and give each receiver the keys of the nodes on the path from the corresponding leaf to the root. With this approach, key storage complexity became logarithmic in terms of the number of receivers, $O(\log n)$.

In [10], which is another milestone of broadcast encryption research, Naor, Naor and Lotspiech proposed two schemes, the complete subtree (CS) and subset difference (SD). The CS scheme was mainly an adaptation of the LKH ideas to BE and has a transmission cost of $O(r \log (n/r))$, $r$ denoting the number of revoked users. The SD scheme decreased the transmission overhead to $O(r)$ at the expense of increasing the key storage to $O(\log^2 n)$. The SD scheme was the most efficient scheme at the time of its proposal, and most of the recent schemes proposed since then are still based on the SD scheme.

The first significant variant of SD was the layered subset difference (LSD) scheme which was proposed by Halevy and Shamir [7]. Optimized LSD has a transmission overhead of $O(\log n \log \log n)$ and key storage of $O(r \log \log n)$. Goodrich, Sun and Tamassia [6] introduced the stratified subset difference (SSD) scheme, which has $O(r \log n / \log \log n)$ transmission overhead and $O(\log n)$ key storage complexity. An analysis of [5, 7, 10] can be found in [8].

The idea of allowing some free riders to get better performance was introduced by Abdalla, Shavitt and Wool [2]. This work was also the first to adapt the key distribution idea of LKH scheme to broadcast encryption. They investigated efficient usage of free riders in depth and developed the basic intuitions about effective assignment of free riders. Ramzan and Woodruff [11] recently proposed an algorithm to optimally choose the set of free riders to be allowed in the CS scheme to minimize the transmission overhead. Their algorithm was based on a dynamic programming approach which decides the free rider assignment in a tree recursively in a bottom-up fashion.

### 1.2 Contributions

In this paper, we study how the transmission cost of an SD scheme can be minimized by an effective placement of a limited number of free riders. The contribution is twofold: First, we give a polynomial-time algorithm which computes the optimal placement for a given number of free riders in an SD scheme. We then propose three heuristic methods which work in a greedy fashion. Experimental results show that significant cost reductions are possible in the SD scheme by these algorithms. They also show that the heuristic methods yield nearly optimal solutions most of the time with a running time dramatically better than that of the optimal algorithm.

### 1.3 Organization

After describing the SD scheme in Section 2, we formalize the problem in Section 3. Section 4 gives the optimal algorithm and Section 5 describes the proposed heuristics for the problem. After presenting the experimental results in Section 6, we conclude the paper in Section 7.

## 2 Subset Difference Scheme

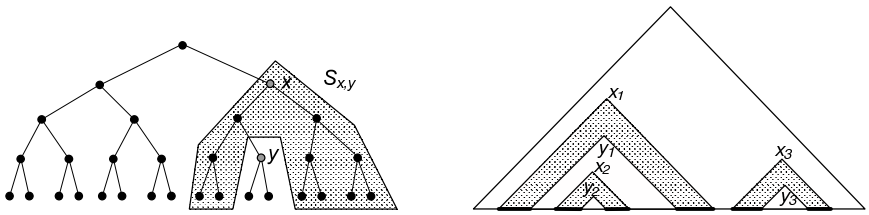
The SD scheme [10], like many other BE schemes, organizes the set of users in the system as leaves of a binary tree. The basic notations regarding this tree are summarized in Table 1. The nodes in the tree are organized into subsets, and an encryption key is assigned to each subset. A user is given the keys of the subsets which he is a member of. The SD scheme is distinguished by the way it defines these subsets: For every non-leaf node $x$, and every descendant $y$ of $x$, a subset is defined as

$$S_{x,y} = \{v \mid v \in T(x) \text{ and } v \notin T(y)\}.$$

The collection of the $S_{x,y}$ subsets is denoted by $\mathcal{S}$. An example subset difference and an example cover are illustrated in Figure 1.

In the broadcast phase of the scheme, to send an encrypted message to a set of privileged users $P$, the center finds a collection $\mathcal{C} \subseteq \mathcal{S}$ that covers $P$,

$$P = \bigcup_{S_{x,y} \in \mathcal{C}} S_{x,y}$$




(a) An SD subset $S_{x,y}$ (b) Cover $C = \{S_{x_1,y_1}, S_{x_2,y_2}, S_{x_3,y_3}\}$

Figure 1: Example subset difference and cover.

Table 1: Notations regarding the SD tree.

| | |
|---|---|
| $L(x)$ : | immediate left child of $x$ |
| $R(x)$ : | immediate right child of $x$ |
| $d(x)$ : | depth of $x$; the distance between $x$ and the root |
| $T(x)$ : | subtree rooted at node $x$ |
| $r(x)$ : | number of revoked users in $T(x)$ |
| $p(x)$ : | number of privileged users in $T(x)$ |

and encrypts a copy of the message encryption key with the key of each subset in $\mathcal{C}$. The encrypted keys are broadcast along with the message. The transmission cost of the broadcast is defined as the number of these encryptions, i.e., the cardinality of the cover $|\mathcal{C}|$.

# 3 Problem Statement

As observed by Abdalla et al. [2], in many cases it may be preferable to allow a limited number of free riders in a BE system in order to reduce the transmission cost. Given the number of free riders that can be tolerated, the question becomes how to utilize this quota most efficiently.

In our treatment, $U$ denotes the set of all receivers, and $P$ and $R = U - P$ denote the set of privileged and revoked receivers respectively, where $n = |U|$, $p = |P|$, $r = |R|$. We denote the tree of all users in the system by $\mathcal{T}$. The free rider quota allowed is denoted by $f$, and $c_f$ denotes the *free rider ratio* $f/p$. The problem is to find a cover $\mathcal{C} \subseteq \mathcal{S}$, $P \subseteq \bigcup_{S_{x,y} \in \mathcal{C}} S_{x,y}$ with $\left| \bigcup_{S_{x,y} \in \mathcal{C}} S_{x,y} - P \right| \leq f$, such that $|\mathcal{C}|$ is minimum.

**Definition 1 (i-point, e-point)** *We call a node $x$ an* inclusion point (i-point) *and $y$ an* exclusion point (e-point) *in an SD configuration where $S_{x,y}$ is in the cover $\mathcal{C}$.*

**Definition 2 (meeting point)** *A node $x$ is called a* meeting point *if both $T(L(x))$ and $T(R(x))$ contain revoked leaves, or if $x$ itself is a revoked leaf.*

A "meeting point" is a point where a branch occurs in the Steiner tree induced by the revoked users in $\mathcal{T}$ (i.e., the minimum subtree in $\mathcal{T}$ that covers all revoked leaves). As in other works [10, 7, 11], this Steiner tree is of particular interest for the optimization algorithms we will discuss. We will denote the highest meeting points in the left and right subtrees of a node $x$ in this tree, i.e., the "meeting-point children" of $x$, by $L_{mp}(x)$ and $R_{mp}(x)$ respectively.

# 4 Optimal Algorithm

In this section, we describe a dynamic programming solution for the SD optimization problem with free riders. The approach is based on the dynamic programming approach of Ramzan and Woodruff [11] for the CS scheme. However, a completely different formulation is needed here due to the complicated relationship between the recursive subproblems in the SD scheme. For the same reason, the approximation algorithm of [11] is also not applicable.

Let $(x; f_x)$ denote the problem instance where exactly $f_x$ free riders are to be placed in $T(x)$. Let $Cost(x, f_x)$ denote the cost of the optimal solution to this problem. Let the left and right meeting-point children of $x$ be $y = L_{mp}(x)$ and $z = R_{mp}(x)$. Consider the case where $f_y$ of the free riders are to be assigned under $y$ and $f_z = f_x - f_y$ of them are to be assigned under $z$. Then, as proven in Section 4.1, the optimal cost for this partition can be expressed in terms of the optimal solutions of $(y; f_y)$ and $(z; f_z)$ as

$$Cost(y, f_y) + Cost(z, f_z) + C_l + C_r, \tag{1}$$

where $C_l$ denotes the additional cost of covering the path between $x$ and $y$ (by addition of either $S_{x,y}$ or $S_{L(x),y}$, as we explain in detail below) and $C_r$ denotes its counterpart between $x$ and $z$. Accordingly, the cost of the optimal solution to the problem $(x; f_x)$ can be expressed as

$$Cost(x, f_x) = \min_{\substack{f_y, f_z \geq 0 \\ f_y + f_z = f_x}} \{Cost(y, f_y) + Cost(z, f_z) + C_l + C_r\}. \tag{2}$$

Now consider $C_l$, the cost of the subset that will be added between $x$ and $y$. First of all, if $f_y = r(y)$, the subtree $T(y)$ and consequently the whole left subtree of $x$ will be privileged, and no subsets will be needed on the left side of $x$.

Given that $T(y)$ is not fully privileged, $S_{x,y}$ will be added to the cover iff $f_z = r(z)$; i.e., iff the right subtree of $x$ is fully privileged.

Given that $T(z)$ is not fully privileged either (i.e., $f_z < r(z)$), the only possible addition on the left side of $x$ is $S_{L(x),y}$, which will take place iff $L(x) \neq y$ (i.e., $y$ is not the immediate left child of $x$).

Addition of $S_{x,y}$ or $S_{L(x),y}$ to the cover may or may not bring an additional cost. If $y$ is an i-point in the optimal solution to $(y; f_y)$, the new set will be merged with the existing set under $y$, and again we will have $C_l = 0$.

Hence the value of $C_l$ is determined as

$$C_l = \begin{cases} 1, & \text{if } f_y < r(y) \text{ and } (f_z = r(z) \text{ or } d(y) - d(x) \geq 2) \text{ and } y \text{ is not an i-point} \\ 0, & \text{otherwise} \end{cases}$$

The value of $C_r$ is determined similarly.

If there is more than one solution that give the minimum cost at (2), the one that makes $x$ an i-point is selected for the possibility of a later merger.

## 4.1 Optimal Substructure Property

Theorem 3 below states the optimal substructure property of the SD optimization with free riders problem.

**Theorem 3** *Let $x$ be a meeting point in an SD tree $\mathcal{T}$, and $y = L_{mp}(x)$ and $z = R_{mp}(x)$. Consider the problem of placing $f_x$ free riders under $x$ optimally where $f_y$ of them are to be placed under $y$. An optimal solution to this problem exists which is based on the optimal solutions of $(y; f_y)$ and $(z; f_z)$, where $f_z = f_x - f_y$.*

**Proof** Assume to the contrary that the optimal solution to the problem at $x$ gives a suboptimal configuration at either $y$ or $z$ (w.l.o.g., assume it is suboptimal at $y$); and assume no equivalent solution exists that is based on some optimal solutions at $y$ and $z$. Let $cost'_y$ denote the cost of the suboptimal configuration at $T(y)$ induced by the optimal solution at $x$. Similarly, let $cost'_z$, $C'_l$, and $C'_r$ denote the costs it induces at subtree $T(z)$, and on the paths $x$-$y$ and $x$-$z$ respectively. Let $cost_y$ and $cost_z$ be the cost of the optimal solutions of $(y; f_y)$ and $(z; f_z)$, and $C_l$ and $C_r$ denote the associated costs on the paths $x$-$y$ and $x$-$z$ in the solution to $(x; f_x)$ based on these optimal solutions at $y$ and $z$. Hence, we have

$$cost'_y + cost'_z + C'_l + C'_r < cost_y + cost_z + C_l + C_r \quad (3)$$

$$cost'_y > cost_y \quad (4)$$

$$cost'_z \geq cost_z. \quad (5)$$

Given that $C_l$ and $C_r$ are either 0 or 1, the situation above is possible only with $C_l = C_r = 1$ and $C'_l = C'_r = 0$. The case $C_l = C_r = 1$ is possible only when (i) $T(y)$ and $T(z)$ are not fully privileged; (ii) $y$ and $z$ are not an immediate child of $x$; and (iii) $y$ and $z$ are not i-points in the optimal solutions of $(y; f_y)$ and $(z; f_z)$. Under conditions (i) and (ii), the assumption that $C'_r = 0$ is possible only when $z$ is an i-point in the corresponding solution in $T(z)$. Given that $z$ was not an i-point in the optimal solution of $(z; f_z)$, this implies $cost'_z > cost_z$. Therefore,

$$cost'_y + cost'_z + C'_l + C'_r \geq cost_y + cost_z + C_l + C_r.$$

## 4.2 Algorithm OptimalAssign

**Algorithm 1** OptimalAssign $(\mathcal{T}, P, f)$

```
1:  MP ← FindMeetingPoints(R)
2:  for i = 1 to r do
3:     x ← MP[i]
4:     C_x[0], C_x[1], I_x[0], I_x[1] ← 0
5:  for i = r + 1 to 2r − 1 do
6:     x ← MP[i]; y ← L_mp(x); z ← R_mp(x)
7:     for f_x = 0 to min(r(x), f) do
8:        C_x[f_x] ← ∞
9:        for f_y = max(f_x − r(z), 0) to min(r(y), f_x) do
10:          f_z ← f_x − f_y
11:          tcost ← C_y[f_y] + C_z[f_z] + C_l + C_r
12:          if tcost < C_x[f_x] or (tcost = C_x[f_x] and (r(y) = f_y or r(z) = f_z))
             then
13:             C_x[f_x] ← tcost
14:             L_x[f_x] ← f_y
15:             I_x[f_x] ← 0
16:             if f_y = r(y) or f_z = r(z) then
17:                I_x[f_x] ← 1
18: root ← MP[2r − 1]
19: (result, f_act) ← FindCost(root)
20: F ← MarkFreeRiders(root, f_act)
21: SDExactAssign(T, P ∪ F)
```

Algorithm 1 shows the optimal algorithm based on the dynamic programming formulation given in (2). The $MP$ array, which is initialized on line 1, contains a list of the meeting points in $\mathcal{T}$. This array is generated by the FindMeetingPoints procedure such that a meeting point is always listed before its parent. Hence as the array is processed in order, the program proceeds from the leaves towards the root. In the course of the algorithm, a two dimensional cost array $C_x[f_x]$ is filled in a bottom-up fashion where a cell $[x, f_x]$ stores the cost of the optimal solution for the subtree of $x$ when $f_x$ free riders are used.

In addition to the cost array, the arrays $I_x$ and $L_x$ are used to maintain the critical information regarding the optimal solution obtained for each problem instance $(x; f_x)$. In the algorithm, $I_x[f_x]$ holds whether $x$ is an i-point in that optimal solution and $L_x[f_x]$ holds how many of the $f_x$ free riders in that optimal solution are assigned to the left subtree of $x$.

The main body of the algorithm OptimalAssign consists of the three nested loops between lines 5–17. The first for loop on line 5 iterates $r - 1$ times; the second loop on line 7 iterates $\min(r(x), f)$ times; and the last one on line 9 iterates $O(\min(r(y), f))$ times. Hence, a straightforward analysis gives the time complexity of the algorithm as $O(rf^2)$. However, as the following theorem proves, a tighter bound can be found as $O(rf + r \log\log n)$. The proof is along

**Algorithm 2** FINDCOST ($root$)

```
1: result ← ∞;
2: for f_root ← 0 to f do
3:    rcost ← C_root[f_root]
4:    if d(root) ≠ 0 then
5:       if I_root[f_root] ≠ 1 then
6:          rcost ← rcost + 1
7:    if result < rcost then
8:       result ← rcost
9:       f_act ← f_root
10: return (result, f_act)
```

**Algorithm 3** MARKFREERIDERS $(x, f_x)$

```
1: F ← ∅
2: if x is not a leaf then
3:    f_y = L_x[f_x]; f_z = f_x − f_y
4:    MarkFreeRiders(L(x), f_y)
5:    MarkFreeRiders(R(x), f_z)
6: else
7:    if f_x = 1 then
8:       F ← F ∪ {x}
9: return F
```

the same lines as that of the dynamic programming algorithm given for the CS scheme in [11].

**Theorem 4** *The time complexity of the algorithm* OPTIMALASSIGN *is* $O(rf + r \log\log n)$.

**Proof** Let $iMP$ denote the set of internal meeting points in $\mathcal{T}$. For a meeting point $x \in iMP$, we will use $y$ and $z$ to denote $L_{mp}(x)$ and $R_{mp}(x)$ such that $r(y) \leq r(z)$. Then, the total complexity of the three nested loops on lines 5–17 is bounded by

$$t = \sum_{x \in iMP} \min(r(x), f) \cdot \min(r(y), f). \tag{6}$$

The terms that contribute to this summation will be analyzed in three classes:

1. $x \in iMP$ such that $r(y), r(z) < f$.
2. $x \in iMP$ such that $r(y) \leq f < r(z)$.
3. $x \in iMP$ such that $f \leq r(y), r(z)$.

We will denote these classes by $MP_1$, $MP_2$, $MP_3$, and their contributions to summation (6) by $t_1$, $t_2$, $t_3$, respectively.

First consider $MP_1$ and $t_1$:

$$\begin{aligned} t_1 &= \sum_{x \in MP_1} r(x)r(y) \\ &= \sum_{x \in MP_1} r(y)r(y) + \sum_{x \in MP_1} r(z)r(y) \end{aligned} \tag{7}$$

Let $t_1'$ and $t_1''$ denote the first and the second halves of summation (7). Since, by definition, $r(y) \leq r(z)$, we have $t_1' \leq t_1''$, and therefore, $t_1 \leq 2t_1''$.

To compute a bound on $t_1''$, we will define a formal variable $X_u$ for each revoked user $u$ and set all of these formal variables to 1. By using these variables, we can write $r(y) = \sum_{u \in R \cap T(y)} X_u$ and $r(z) = \sum_{u \in R \cap T(z)} X_u$; hence,

$$r(z)r(y) = \sum_{\substack{u \in R \cap T(y) \\ v \in R \cap T(z)}} X_u X_v,$$

where every $X_i$ equals 1.

Now consider the question that how many monomials $X_u X_v$ a particular revoked user $u$ contributes to the summation $t''$. Let $\mathcal{T}'$ denote the Steiner tree consisting of the meeting points in $\mathcal{T}$, where a meeting point $x$ and its meeting-point children $L_{mp}(x)$ and $R_{mp}(x)$ are linked directly. Let $x$ be the highest ancestor of $u$ in $\mathcal{T}'$ that is in $MP_1$. Consider the path $u = u_0, u_1, \ldots, u_k = x$ in $\mathcal{T}'$. Let $v_i$ be the sibling of $u_i$ for $0 \leq i < k$. Since $T(v_i)$ and $T(v_j)$ are disjoint for all $i \neq j$ there are $\sum_{i=0}^{k-1} |r(v_i)|$ monomials containing $X_u$ and each of them has coefficient 1. So the number of monomials containing $X_u$ can be no more that $2f$ since $x \in MP_1$ and $T(x)$ contains at most $2f$ revoked users. Given that there are $r$ revoked users in total, we have $t_1'' = O(rf)$, and consequently, $t_1 = O(rf)$.

Second, consider $MP_2$ and $t_2$:

$$\begin{aligned} t_2 &= \sum_{x \in MP_2} \min(r(x), f) \cdot \min(r(y), f) \\ &= \sum_{x \in MP_2} fr(y) \end{aligned}$$

Note that any $x \in MP_2$ cannot be a descendant of any other $x' \in MP_2$; hence the $T(y)$, $T(y')$ subtrees are disjoint for any distinct $x, x' \in MP_2$. Therefore, we have

$$t_2 = f \sum_{x \in MP_2} r(y) \leq rf.$$

Third and the last, consider $MP_3$ and $t_3$. Consider the subtree $\mathcal{T}'' \subset \mathcal{T}'$ consisting only of the meeting points in $MP_3$ and their left and right children. Since there are $r$ revoked users in total, there can be at most $r/f$ leaves in $\mathcal{T}''$. So, the number of the meeting points in $MP_3$ is no more than $r/f - 1$. Note that the contribution of a meeting point in $MP_3$ to $t_3$ is $f^2$; hence $t_3 = f^2 O(r/f) = O(rf)$.

Since each of $t_1$, $t_2$, and $t_3$ are $O(rf)$, we have $t = O(rf)$. Besides, finding the meeting points at the beginning of the algorithm takes $O(r \log \log n)$ time [11]. Hence, the overall time complexity of the algorithm OPTIMALASSIGN is $O(rf + r \log \log n)$.

# 5 Greedy Heuristics

When a faster solution is needed, a heuristic algorithm which gives nearly optimal solutions in a shorter time can be preferred. In this section we describe three heuristic methods for this purpose, two greedy algorithms and a third combined method, which return near-optimal results with a running time significantly faster than that of the optimal algorithm.

## 5.1 Top-Down Heuristic

The first heuristic searches the user tree in a top-down fashion to identify the $S_{x,y}$ subsets to cover a given receiver set $P$, such that each subset taken satisfies in itself the free rider ratio $c_f = f/p$.

Note that an SD tree cannot be searched greedily by just looking at single nodes since the $S_{x,y}$ subsets are defined by two nodes having a descendant-ascendant relationship. We define an exclusion point $e(x)$ for every node $x$ to be the descendant of $x$ with the largest subtree under it that is completely revoked. The TOPDOWNASSIGN heuristic first calls the FINDEPOINTS procedure, which identifies $e(x)$ for every $x$ recursively beginning from the root. Then TOPDOWNCOVER is called, which searches the tree top to bottom for subsets that satisfy the free rider ratio $c_f$.

TOPDOWNCOVER($x$) takes $S_{x,e(x)}$ into the cover if it satisfies the free rider ratio. Else, if $x$ is a meeting point, the procedure is called recursively on $L(x)$ and $R(x)$. If $x$ is not a meeting point, then a subset that covers all privileged descendants of $x$ until the first meeting point is added to the cover, and the procedure is repeated beginning from that meeting point.

**Algorithm 4** TOPDOWNASSIGN($\mathcal{T}, P, f$)

1: FINDEPOINTS(*root*)
2: $c_f \leftarrow f/p$
3: $\mathcal{C} \leftarrow \emptyset$
4: TOPDOWNCOVER(*root*)

The TOPDOWNASSIGN heuristic has two main subroutines; FINDEPOINTS and TOPDOWNCOVER. Both subroutines are recursive methods called once for each meeting point, and do a constant amount of work at each call, hence have a complexity of $O(r)$. The complexity of the algorithm also includes the cost of finding meeting points which is $O(r \log n)$. Hence, the overall time complexity of TOPDOWNASSIGN is $O(r \log n)$.

**Algorithm 5** FINDEPOINTS($x$)

```
1: if r(x) > 0 then
2:    if p(x) = 0 then
3:       e(x) ← x
4:    else
5:       y ← FINDEPOINTS(L(x))
6:       z ← FINDEPOINTS(R(x))
7:       if r(y) > r(z) then
8:          e(x) ← y
9:       else
10:         e(x) ← z
11:   return e(x)
12: else
13:   return null
```

**Algorithm 6** TOPDOWNCOVER($x$)

```
1: if (r(x) − r(e(x)))/(p(x) − p(e(x))) ≤ c_f then
2:    C ← C ∪ {S_{x,e(x)}}
3: else
4:    if r(L(x)) > 0 and r(R(x)) > 0 then
5:       TOPDOWNCOVER(L(x))
6:       TOPDOWNCOVER(R(x))
7:    else
8:       if r(R(x)) = 0 then
9:          C ← C ∪ {S_{x,L_mp(x)}}
10:         TOPDOWNCOVER(L_mp(x))
11:      if r(L(x)) = 0 then
12:         C ← C ∪ {S_{x,R_mp(x)}}
13:         TOPDOWNCOVER(R_mp(x))
```

## 5.2 Bottom-Up Heuristic

The free rider quota can be utilized more efficiently by a targeted free rider placement heuristic that places the free riders on an existing solution to merge the subsets in the cover $\mathcal{C}$. One can remove an existing $S_{x,y}$ subset from $\mathcal{C}$ by saturating $T(y)$ with free riders. Then $T(x)$ will become fully privileged and has to be covered. Accordingly, the subset $S_{parent(x),sibling(x)}$ is temporarily added to the cover, and it is determined whether it can be merged with any other subsets or not. There are three possibilities regarding the reduction in the cover size $|\mathcal{C}|$:

- **0:** There will be no reduction if the subset $S_{parent(x),sibling(x)}$ cannot be merged with any other subset. This happens when neither $parent(x)$ is the e-point nor $sibling(x)$ is the i-point of any other subset in $\mathcal{C}$.
- **1:** A reduction of 1 will be obtained when the subset $S_{parent(x),sibling(x)}$ can only be merged with either $S_{x',parent(x)}$ or $S_{sibling(x),y'}$ for some $x'$ or $y'$.
- **2:** As the best case, a reduction of 2 will be obtained when $S_{parent(x),sibling(x)}$ can be merged with both $S_{x',parent(x)}$ and $S_{sibling(x),y'}$, for some $x'$, $y'$.

To decide which subset to remove next, the BottomUpAssign heuristic uses the *rate of return*, defined as the reduction in the cover size divided by the number of free riders needed. The heuristic dynamically maintains a priority queue $SL$ of subsets in the current cover ordered according to their rate of return. Whenever a subset is to be removed, the first one in the queue is selected.

**Algorithm 7** BottomUpAssign$(\mathcal{T}, P, f)$

```
1: C ← SDExactAssign(T, P)
2: SL ← GetPQ(C, f)
3: while SL ≠ ∅ do
4:    repeat
5:       (x, y) ←ExtractFirst(SL)
6:    until r(y) ≤ f
7:    C ← C − {S_{x,y}}
8:    Saturate(y)
9:    (x_new, y_new) ← Merge(C, SL, x)
10:   C ← C ∪ {S_{x_new,y_new}}
11:   Insert(SL, S_{x_new,y_new})
12:   f ← f − r(y)
```

The GetPQ procedure produces the priority queue $SL$ of $S_{x,y}$ subsets with $r(y) \leq f$, ordered according to their rate of return. The ExtractFirst procedure extracts the first subset $S_{x,y}$ in $SL$ and returns the corresponding indices. The Saturate procedure updates the $r$ and rate of return values of all ascendants of $y$, rearranging $SL$ accordingly.

**Algorithm 8** MERGE($\mathcal{C}, SL, x$)

**if** $S_{x',parent(x)} \in \mathcal{C}$ for some $x'$ **then**
  $x_{new} \leftarrow x'$
  $\mathcal{C} \leftarrow \mathcal{C} - \{S_{x',parent(x)}\}$
  REMOVE($SL, S_{x',parent(x)}$)
**else**
  $x_{new} \leftarrow parent(x)$
**if** $S_{sibling(x),y'} \in \mathcal{C}$ for some $y'$ **then**
  $y_{new} \leftarrow y'$
  $\mathcal{C} \leftarrow \mathcal{C} - \{S_{sibling(x),y'}\}$
  REMOVE($SL, S_{sibling(x),y'}$)
**else**
  $y_{new} \leftarrow sibling(x)$
**return** $(x_{new}, y_{new})$

Regarding the time complexity of the BOTTOMUPASSIGN heuristic, finding the initial exact SD assignment takes $O(r \log n)$ time. Then creation of the priority queue $SL$ takes $O(r \log r)$ time. In the while loop, the EXTRACTFIRST routine is called $O(r)$ times in total, among which at most $f$ lead to a set merger. Those that don't lead to a merger will be completed in $O(r \log r)$ time in total. For those that do, a run of INSERT, REMOVE, and SATURATE may be needed per merger. INSERT and REMOVE take $O(\log r)$ time. SATURATE includes $O(\log n)$ decrease key operations, each of which may take $O(\log r)$ or $O(1)$ time depending on whether a binary or Fibonacci heap is used for implementing $SL$, making the total cost of the set merger operations $O(f \log n \log r)$ or $O(f \log n)$ accordingly. Therefore, the overall complexity of BOTTOMUPASSIGN is $O(r \log n + f \log n \log r)$ with a binary heap implementation and $O(r \log n)$ with a Fibonacci heap implementation of the priority queue $SL$.

### 5.3 Hybrid Heuristic

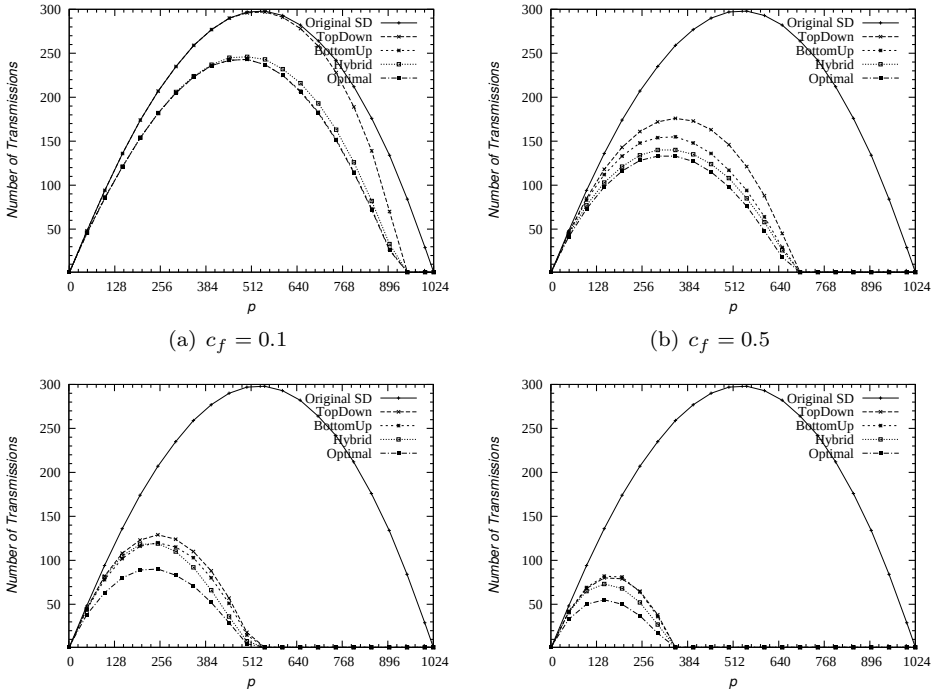
The running time of the BOTTOMUPASSIGN heuristic increases significantly when the amount of the free rider quota to be placed is high. This problem can be solved by using the TOPDOWNASSIGN procedure to obtain an initial configuration and running BOTTOMUPASSIGN on top of it, instead of starting BOTTOMUPASSIGN with an exact SD cover and placing all free riders one by one. This combined method, which we call HYBRIDASSIGN, returns near-optimal solutions significantly faster than the original BOTTOMUPASSIGN.

## 6 Experimental Results

We tested the practical performance of the algorithms in a series of simulation experiments, conducted with the parameters $n = 1024$, $1 \leq p \leq 1024$, and $0 \leq c_f \leq 2$. We summarize the results in this section. Each data point in the

plots is averaged over 50000 runs where the revoked users are randomly selected among the user set.

Figures 2 and 3 compare the transmission costs obtained by the proposed algorithms against that of the basic SD scheme. Figure 2 presents the results according to the privileged set size $p$ for a set of selected $c_f$ values. Figure 3 presents the results according to the free rider ratio $c_f$.



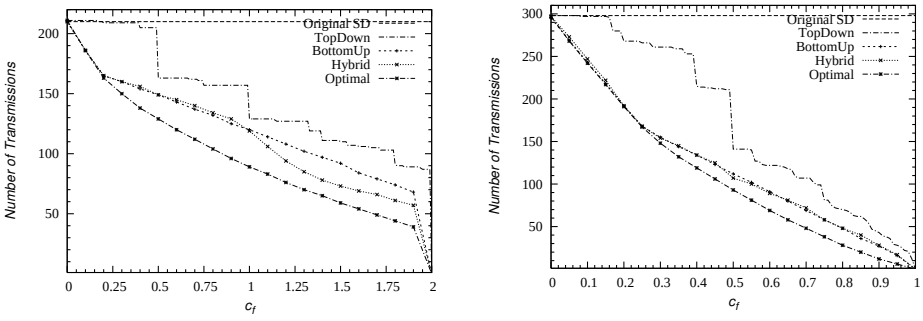

(a) $c_f = 0.1$ (b) $c_f = 0.5$

(c) $c_f = 1.0$ (d) $c_f = 2.0$

Figure 2: Transmission costs of the algorithms with respect to $p$.

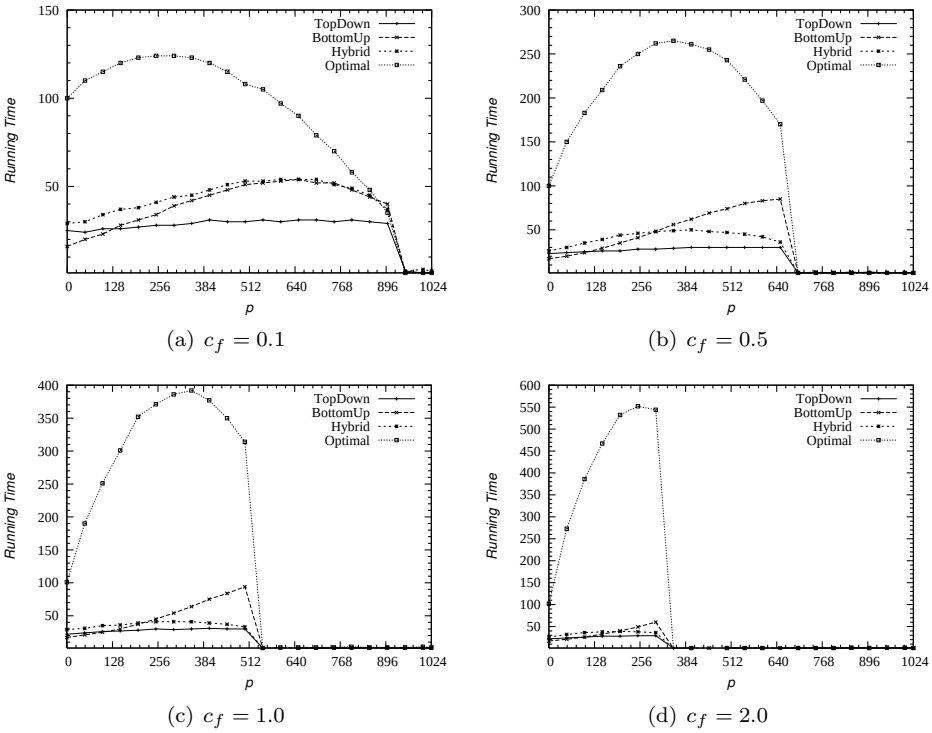
The results show that significant gains are possible by the proposed algorithms. With a limited free rider ratio such as 0.1, 20% or more reduction can be obtained; and when larger values of $c_f$ are tolerable, a reduction of 80% or more is possible. The experiments also show that the results returned by the HybridAssign heuristic are usually very close to the results obtained by the optimal algorithm.

Figure 4 compares the running time of our algorithms. The results show that HybridAssign turns out to have the best cost-benefit performance among the heuristic methods. Its running time is only slightly more than that of TopDownAssign while its performance matches that of BottomUpAssign and sometimes approaches to that of the optimal algorithm.




(a) $p = 256$ (b) $p = 512$

Figure 3: Transmission costs of the algorithms with respect to $c_f$.




(a) $c_f = 0.1$ (b) $c_f = 0.5$

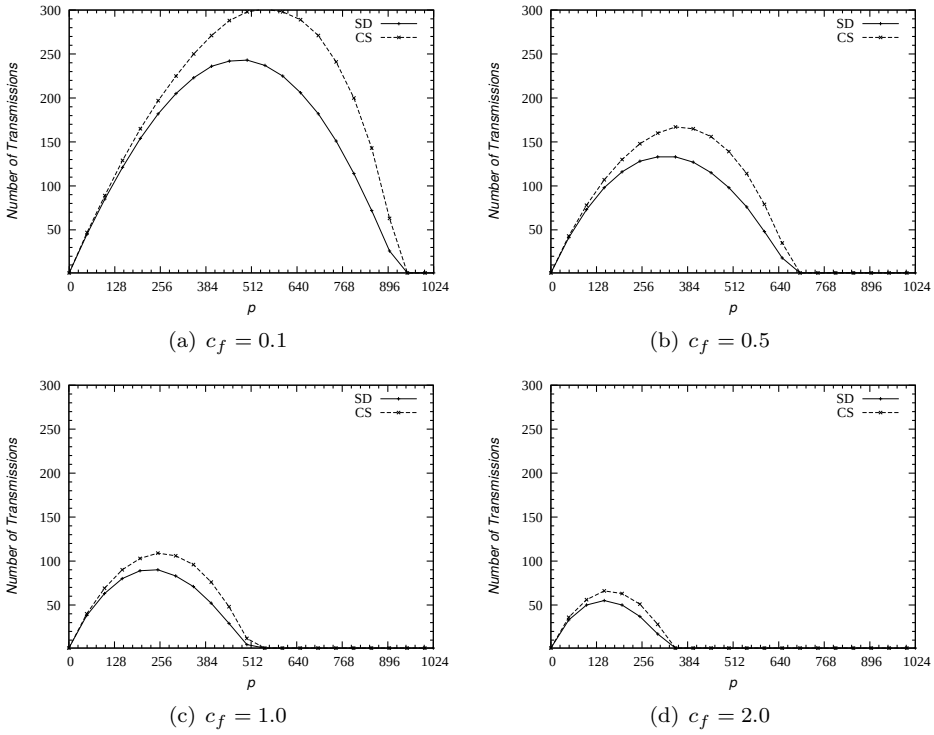
(c) $c_f = 1.0$ (d) $c_f = 2.0$

Figure 4: Execution time of the algorithms in seconds. The figures are the total time of the 50000 runs taken for each data point.

## 6.1 Comparison with the CS Scheme

An optimal free rider assignment algorithm for the CS scheme was given by Ramzan and Woodruff [11]. We also implemented this algorithm and compared

it to our optimal algorithm for the SD scheme. Figure 5 compares the performance of the two optimal algorithms in terms of the transmission cost. The results show that, with the same number of free riders allowed, the SD scheme can give a transmission cost 20% less than that of the CS scheme.




(a) $c_f = 0.1$ (b) $c_f = 0.5$

(c) $c_f = 1.0$ (d) $c_f = 2.0$

Figure 5: Transmission costs obtained by the optimal algorithms for the CS and the SD schemes.

# 7 Conclusion

The SD scheme is one of the most efficient BE schemes today. In this paper, we studied the problem of improving the performance of an SD scheme by allowing a limited number of free riders in the system. We first proposed an optimal algorithm based on a dynamic programming approach which finds the best free rider placement that leads to the minimum transmission overhead. Subsequently, we proposed three heuristics for the same problem that return near-optimal solutions with a faster running time. The TopDownAssign heuristic works extremely fast, but it may not utilize all the available free rider quota, or it may spend a large amount of it fast and carelessly, possibly missing configurations that are more efficient. These drawbacks were solved in

the BottomUpAssign heuristic, which uses a targeted placement approach, placing the free riders slowly and carefully, and using all the available quota. However, this procedure gets slower as the free rider quota to be placed increases. Noting the different advantages and drawbacks of the two procedures, we offered a third heuristic, HybridAssign, that combines the advantages of the two approaches.

The experimental results show that the optimal placement algorithm and the three heuristics proposed provide significant reductions in the transmission cost of the SD scheme.

Besides the basic SD scheme, these algorithms can also be applied to its variants such as LSD [7] and SSD [6]. These variants differ from the basic SD in the way key generation is done, but they are exactly the same as far as cover finding is concerned. Hence, the systems based on these SD variants can benefit equally from the proposed algorithms.

# References


[1] AACS - Advanced Access Content System, http://www.aacsla.com, 2007.

[2] M. Abdalla, Y. Shavitt, and A. Wool. Key management for restricted multicast using broadcast encryption. *IEEE/ACM Trans. Networking*, 8(4):443–454, 2000.

[3] W. Aiello, S. Lodha, and R. Ostrovsky. Fast digital identity revocation. In *CRYPTO'98*, volume 1462 of *LNCS*, pages 137–152. Springer-Verlag, 1998.

[4] S. Berkovits. How to broadcast a secret. In *EUROCRYPT'91*, volume 547 of *LNCS*, pages 535–541. Springer-Verlag, 1991.

[5] A. Fiat and M. Naor. Broadcast encryption. In *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer-Verlag, 1993.

[6] M. T. Goodrich, J. Z. Sun, and R. Tamassia. Efficient tree based revocation in groups of low-state devices. In *CRYPTO'04*, volume 3152 of *LNCS*, pages 511–527. Springer-Verlag, 2004.

[7] D. Halevy and A. Shamir. The LSD broadcast encryption scheme. In *CRYPTO'02*, volume 2442 of *LNCS*, pages 47–60, London, UK, 2002. Springer-Verlag.

[8] J. Horwitz. A survey of broadcast encryption, 2003. Manuscript.

[9] J. Lotspiech, S. Nusser, and F. Pestoni. Broadcast encryption's bright future. *Computer*, 35:57–63, 2002.

[10] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO'01*, volume 2139 of *LNCS*, pages 41–62. Springer-Verlag, 2001.

[11] Z. Ramzan and D. Woodruff. Fast algorithms for the free riders problem in broadcast encryption. In *CRYPTO'06*, volume 4117 of *LNCS*, pages 308–325. Springer-Verlag, 2006.

[12] C. B. S. Traw. Protecting digital content within the home. *Computer*, 34:42–47, 2001.

[13] D. M. Wallner, E. J. Harder, and R. C. Agee. Key management for multicast: Issues and architectures, 1999. Internet draft.

[14] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communication using key graphs. In *SIGCOMM'98*, pages 68–79, September 1998.