

OPTIMAL BROADCAST ENCRYPTION WITH FREE RIDERS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Murat Ak

September, 2006

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Ali Aydın Selçuk(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. İbrahim Körpeoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Alper Şen

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

OPTIMAL BROADCAST ENCRYPTION WITH FREE RIDERS

Murat Ak

M.S. in Computer Engineering

Supervisor: Assist. Prof. Dr. Ali Aydın Selçuk

September, 2006

Broadcast encryption schemes allow a center to broadcast encrypted messages so that each particular message can only be decrypted by a set of privileged receivers designated for it. The standard technique is to distribute keys to all receivers at the beginning and to use only the necessary keys to encrypt the message for each particular broadcast. The number of encryptions needed constitutes the transmission cost in broadcast encryption schemes. The most efficient scheme in terms of transmission cost published so far is the subset difference (SD) scheme. It is possible to reduce the transmission overhead of a broadcast encryption scheme by allowing a number of free riders to be able to decrypt the message although they are not privileged. However, unless the free riders are chosen cleverly, the cost may even increase. In this thesis, we deal with the problem of choosing a given number of free riders effectively to reduce the transmission overhead in SD scheme. We first present three greedy algorithms. First algorithm has a fast execution, the second one is slower but has lower transmission overhead, and the last one has a trade-off between running time and transmission cost. Then we present two algorithms which find the minimum transmission overhead possible given a free rider quota. The experiments we conduct show that the transmission cost can be significantly reduced.

Keywords: Broadcast Encryption, Free Riders.

ÖZET

MUAF KULLANICILARLA EN UYGUN YAYIN ŞİFRELENMESİ

Murat Ak
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Assist. Prof. Dr. Ali Aydın Selçuk
Eylül, 2006

Yayın şifreleme planları bir merkezin şifreli mesajlar yayınlamasını sağlar. Öyle ki her bir mesajı sadece o mesaj için belirlenmiş imtiyazlı bir grup deşifre edebilir. Standart teknik en başta tüm alıcılara anahtarlar dağıtıp her bir yayında mesajı sadece gerekli anahtarlarla şifrelemektir. Yapılan şifreleme sayısı gönderim masrafını oluşturmaktadır. Bu masraf bakımından şu ana kadar yayınlanmış en iyi plan Altküme Farkı (AF) planıdır. Bir yayın şifreleme planında imtiyazlı olmayan bir takım alıcıların da yayını deşifre edebilmesine izin verilmesi suretiyle gönderim masrafının azaltılması mümkündür. Fakat bunun için muaf kullanıcıların akıllıca seçilmesi gerekmektedir. Bu tez AF planında gönderim masrafını azaltmak için izin verilen sayıda muaf kullanıcının etkin olacak şekilde seçilmesi problemini hedef almaktadır. Öncelikle üç adet haris algoritma sunuyoruz. Bunlardan birincisi hızlı çalışma süresine, ikincisi daha yavaş çalışmasına karşın daha az gönderim masrafına sahiptir. Sonuncusu ise bu iki unsur arasında dengeyi sağlamaktadır. Daha sonra belli bir muaf kullanıcı kotası verildiğinde asgari gönderim masrafına ulaşan iki algoritma sunuyoruz. Yürütmüş olduğumuz deneyler göstermektedir ki gönderim masrafı büyük ölçüde azaltılabilmektedir.

Anahtar sözcükler: Yayın Şifreleme, Muaf Kullanıcı.

Acknowledgement

I would like to express my gratitude to Dr. Ali Aydın Selçuk for offering his valuable time and support as my supervisor. I have learned a lot from his suggestions during this research. He also helped me to keep my interest all along.

I am grateful to Dr. İbrahim Körpeoğlu and Dr. Alper Şen for accepting to read and review this thesis.

I am also indebted to Kamer Kaya who helped me with the technical aspects of this research and gave inspirations for some of the algorithms.

I would like to thank the other members of Security Reading Group (SRG) who have been very gentle and friendly since the beginning.

I would like to express my gratitude and appreciation to my parents for their infinite love, support and guidance.

I acknowledge that TUBITAK (The Scientific and Technical Research Council of Turkey) supported this study under MSc. Scholarship program.

Anneme ve Babama.

Contents

1	Introduction	1
1.1	Related Work	3
1.2	Problem Definition	4
2	Free Rider Problem and SD Scheme	6
2.1	Notation and Subset Cover Framework	6
2.2	Free Rider Assignment Problem	8
2.2.1	Tree Scheme and f-Cover Algorithm	11
2.2.2	The Complete Subtree Scheme	11
2.2.3	CS with Optimal Free Rider Assignment	12
2.3	The Subset Difference Scheme	12
2.4	Definitions	14
3	Greedy Algorithms	18
3.1	Top-Down Algorithm	18
3.2	Bottom-Up Algorithm	20

3.3	Hybrid Algorithm	25
4	Optimal Free Rider Assignment for SD	26
4.1	The Optimization Algorithms	26
4.1.1	The Basic Algorithm	27
4.1.2	Improved Algorithm	32
5	Experimental Results	37
5.1	Evaluation Parameters	37
5.2	Implementation Issues	38
5.3	Experiments	39
5.3.1	Transmission Overhead Experiments	39
5.3.2	Running Time Comparisons	43
6	Conclusion and Future Work	45

List of Figures

2.1	Subset Cover Example 1	8
2.2	Subset Cover Example 2	9
2.3	Level ordering	15
2.4	SD Configuration	15
3.1	Removal of a subset	21
3.2	No profit	22
3.3	Profit of 1	22
3.4	Profit of 2	23
5.1	Transmission costs with $f = 2.0$	39
5.2	Transmission costs with $f = 1.6$	40
5.3	Transmission costs with $f = 1.34$	40
5.4	Transmission costs with $f = 1.231$	41
5.5	Transmission costs with $f = 1.143$	41
5.6	Transmission costs with $f = 1.067$	42

List of Tables

2.1	Preliminary Notation	14
5.1	Running Times of the Algorithms	44

Chapter 1

Introduction

Broadcast encryption is the process of sending a message to a set of receivers by encrypting it to guarantee that only a designated subset of the receivers is able to decrypt, preventing the others to decrypt. Idea of broadcast encryption maintains a solution to the problem of secure communication over one way channels, specifically when a center has to communicate with a huge group of receivers. It has many applications but the most popular ones are content protection and media multicast over cable television or internet. Content protection is the act of preventing unauthorized copying of copyrighted CDs, DVDs, etc. In media multicast, broadcast encryption is needed when a cable TV channel wants to restrict access to its programs to the subscribed users.

Broadcast encryption is usually done as follows: Receivers are given some symmetric keys at the beginning. For example, in cable TV scenario, these keys are burned into the set-top boxes in the devices of the customers. When the center is going to send a message, it encrypts the message with a master key, and encrypt this key with each of some selected keys among the ones that was given to the receivers, and broadcasts them together with the encrypted message. Upon a message arrival, a receiver is able to view the message if it has one of the keys that the center used.

The aim of the broadcast encryption schemes is to improve basically two

important evaluation parameters, the number of keys stored per receiver and the number of encryptions that the center do, referred as *key storage* and *transmission overhead* respectively. The complexities are usually in terms of the number of all receivers N , and the number of revoked users, r . These parameters will be explained in detail in Chapter 2.

In broadcast encryption, receivers are modelled as either *stateless* or *dynamic*. When we talk about *stateless receivers*, that means the keys assigned to the receivers do not change once they are given at the beginning. However, keys of *dynamic receivers* are changed while the system is running, which is called *rekeying*. This approach is mostly suitable for only the applications where most of the receivers are privileged. Therefore it is mostly used for rekeying when a key of a receiver is compromised. On the other hand, stateless receivers are appropriate for cable TV applications, because a program may not be subscribed by most of the customers, and since cable TV's has thousands or millions of users, it is infeasible to rekey whenever a user is revoked. In this work, we deal with the stateless receiver case.

The thesis is organized as follows: In Chapter 2, we present the broadcast problem in detail and explain some of the early results. Then we review subset difference scheme which is the current state of the art in broadcast encryption. Then we present the preliminaries, definitions and notations to be used in our algorithms. In Chapter 3, we give three algorithms for the broadcast problem with free riders. Basically, first two algorithms are based on different greedy heuristics, and the last one is obtained by merging them. In Chapter 4, we propose two slightly different algorithms for optimal free rider assignment. Both algorithms are based on a dynamic programming approach. In Chapter 5, we give the experimental results comparing the performances of our algorithms. In chapter 6, we conclude our work by discussing our results.

1.1 Related Work

Two years after Berkovits [1] introduced the idea of broadcast encryption in 1991, Fiat and Naor [2] presented their model which is accepted as the first formal work and milestone in the topic. They introduced the *resiliency* concept, and defined *k-resilient* to be resilient to coalition of at most k 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. The schemes published after this work assumed that perfect resiliency is required and resiliency concept was no longer used. After this work, broadcast encryption became a heavily studied research area. In 1998, Luby and Staddon [3] studied the trade off between the transmission and storage costs and gave combinatorial bounds.

In the late nineties, Wallner et al. [5] and Wong et al. [6] independently proposed logical tree hierarchy (LKH) for secure internet multicast. LKH was not a broadcast encryption scheme at all, 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 to each node of the tree and give a 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, $O(\log N)$, which was called as *log-key storage* restriction later.

In [7], which is the second milestone of broadcast encryption research, Naor, Naor and Lotspiech proposed two schemes, complete subtree (CS) and subset difference (SD). In fact, CS scheme was summarizing the current state of the art before SD scheme. It was actually a specific case of the scheme of Abdalla et al. [4] where redundancy is 0. CS scheme had $O(\log N)$ key storage and $O(r \log N/r)$ transmission cost. What they did in SD scheme was to decrease the transmission overhead to $O(\log r)$ at the expense of increasing the key storage to $O(\log^2 N)$. SD scheme was the best scheme by that time, and most of the new schemes proposed until today are still based on the SD scheme.

The first significant variant of SD was layered subset difference (LSD) which

was proposed by Halevy and Shamir in [8]. Optimized LSD has a transmission overhead of complexity $O(\log N \log \log N)$ and key storage of complexity $O(r \log \log N)$. Later, Goodrich, Sun and Tamassia [10] introduced stratified subset difference (SSD) scheme, which has $O(\log r)$ transmission overhead and $O(\log N)$ key storage complexity. A detailed analysis of [2, 7, 8] can be found in [9].

The idea of allowing some free riders to get better performance, was introduced by Abdalla, Shavitt and Wool [4] in 1999. 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 figured out basic intuition about effective assignment of free riders.

Although allowing free riders is a useful approach, surprisingly, it was not used since [4] until Ramzan and Woodruff [19] recently proposed an algorithm which optimally chooses which revoked users to be allowed to be free riders to get minimal transmission overhead in CS scheme. Their algorithm was based on a dynamic programming approach, which decides the free rider assignment positions in subtrees in a tree recursively in a bottom-up fashion.

1.2 Problem Definition

Most of the work published so far has the following restriction: A message can be decrypted by privileged users but it cannot be decrypted by any other user even by coalitions of them. We call this restriction as the *exact cover* restriction. This level of security is certainly useful for applications which transfer critical data such as the ones used by military or intelligence agencies but especially commercial applications like TV broadcasts do not strictly need such a tight security. In fact, loosening this restriction may be useful by reducing the traffic of the network.

In the literature, *free riders* are defined as the receivers who are not privileged but are able to decrypt the message. Allowing free riders is useful because the transmission costs of the possible privileged receiver subsets are not same and

when the system have the chance to change the privileged subset slightly it will probably find a lower-cost subset which covers the original one and not much larger than it.

The problem we address in this thesis is how to minimize the transmission overhead by cleverly choosing a given number of free riders in subset difference scheme. The same problem was investigated and solved for complete subtree (CS) scheme in [19] by Ramzan and Woodruff. However, the subset difference (SD) scheme is the base scheme for the most efficient schemes today such as LSD [8] and SSD [10]. Therefore, the problem of optimally choosing the free riders should be solved for SD to get the best scheme that allows free riders.

In order to understand the free rider problem and to see its advantage, we need to know the subset structure used by the broadcast encryption schemes. We explain the subset cover framework in Section 2.1. Then we will explain the free rider assignment problem in detail in Section 2.2. Afterwards, we treat the free rider problem for SD scheme.

Chapter 2

Free Rider Assignment Problem and SD Scheme

In this chapter, we first give the preliminary notation and Subset Cover framework. Then we explain the research problem of the thesis. Then we discuss the schemes before SD in detail and we explain the SD scheme. We finish the chapter by giving some definitions needed for our algorithms in the next chapters.

2.1 Notation and Subset Cover Framework

We denote the set of all users by U , with $|U| = N$. The set of revoked users for a particular broadcast is denoted by R , with $|R| = r$. The aim of the center is to send a message M encrypted by some keys to all users such that only the users in $U \setminus R$ can decrypt the ciphertext and get M .

Throughout this thesis, we will stick to the subset cover framework of [7], which formalizes the design process of a broadcast encryption scheme by explicitly defining the common properties and steps of every broadcast encryption scheme. In this framework, a collection $\mathcal{S}$ is defined first.

$$\mathcal{S} = \{S_i | S_i \subseteq U\}$$

Then, every subset S_i is associated with a key K_i and this key is given to all receivers in S_i . Upon a broadcast need, given the revoked user set R , an algorithm COVER finds a set C (for cover) of the subsets such that

$$U \setminus R \subseteq \bigcup S_i \in C$$

Then the message M is encrypted $|C|$ times separately with the keys of the subsets in C . The minimum cardinality C is the optimal cover denoted by C_{opt} . Any algorithm COVER is aimed to find a C as small as possible.

Any broadcast encryption scheme using the formulation of subset cover framework, has to have three components.

- **A Key Distribution Algorithm.** This is the algorithm that defines the subsets. Once the subsets are defined, each subset is assigned a different key.
- **A Covering Algorithm.** First it chooses a session key, K . Then runs the COVER algorithm to find the cover, which is the set of subsets whose union covers the privileged subset of receivers. Finally it encrypts K with the keys of the subsets found by the COVER algorithm. Then, it sends the ID's (i's) of the subsets, encrypted session keys ($E_{K_i}^1(K)$'s) and the message M encrypted with K , ($E_K^2(M)$) as the ciphertext.
- **A Decryption Algorithm.** Upon a message arrival to a receiver u , it finds whether any subsets that includes it is in the cover and if so, it extracts and decrypts $E_{K_u}^1(K)$ with K_u and gets K . Then it decrypts $E_K^2(M)$ with K and gets M .

Note that the third step is straightforward. The first two steps are the important steps. Also note that they are consecutive and every step depends on the previous

ones. In this thesis we stick to the key distribution of SD scheme and deal with only the covering algorithm.

2.2 Free Rider Assignment Problem

In order to understand the essence of the free rider assignment problem, how and why we get to this problem, we must first look at the two basic problems of broadcast encryption schemes. First problem is how to distribute the keys to the receivers at the beginning, and the second one is how to select the keys which will be used in the encryptions for a particular broadcast. We call the first one *key distribution* problem, and the second one *broadcast cover* problem. We explain the problems in details as follows.

In a solution to the key distribution problem, one must be able to cover every possible subset of receivers because the privileged subset may be any of these subsets. For example, consider there are six receivers r_1 through r_6 . Consider we distribute three keys, k_1, k_2, k_3 as follows. We give k_1 to r_1 and r_2 ; k_2 to r_2, r_3 and r_4 ; k_3 to r_4, r_5 and r_6 . (See Figure 2.1). With this distribution, one can cover the subset $\{r_1, r_2\}$ by encrypting the broadcast with only k_1 , or $\{r_2, r_3, r_4, r_5, r_6\}$ by encrypting once with k_2 and once with k_3 . However, one cannot cover the subset $\{r_3, r_4\}$. In a valid key distribution, this situation should be prevented. Actually, the rest is minimizing the number of keys per receiver called *key storage*, which is subject to the *key distribution* problem. At the moment the broadcast cover problem is encountered, keys have been already distributed to the receivers, and a particular broadcast, for which we know exactly which receivers are revoked and which are privileged, is ongoing. The problem is determining how many encryptions are needed and with which keys they should be done. Of course, we

r_1	r_2	r_3	r_4	r_5	r_6
k_1	k_1				
	k_2	k_2	k_2		
			k_3	k_3	k_3

Figure 2.1: Example Subset Cover

r_1	r_2	r_3	r_4	r_5	r_6
	k_1	k_1			
		k_2	k_2		
k_3	k_3	k_3	k_3		

Figure 2.2: Example Subset Cover for Free Rider Demonstration

want minimize the number of encryptions, called *transmission overhead*, which is subject to the *broadcast cover* problem.

These two problems are strongly dependent when free riders are not allowed. We cannot improve one of them without affecting the other, because once we design a solution for the first problem, solution to the second problem becomes deterministic. But once free riders are allowed, transmission cost can be reduced without affecting the key storage. To demonstrate this idea, consider the following example. Let we have six receivers r_1 through r_6 , and let k_1 is given to r_2 and r_3 ; k_2 is given to r_3 and r_4 ; k_3 is given to r_1 , r_2 , r_3 , and r_4 ; and k_4 is given to r_5 and r_6 . See Figure 2.2. Now suppose that we have three privileged users, r_2 , r_3 , and r_4 . We can cover these receivers by using keys k_1 and k_2 , and have a transmission overhead of 2. However, if we allow 1 free rider, we can use it for r_1 and use only k_3 which leads us to a transmission overhead of 1. Note that we placed the free rider not randomly but cleverly. With a random placement, the probability of success would be $1/3$. Here, we see that assigning the only free rider to r_1 is obviously the optimum solution. However, when we have lots of receivers and lots of free riders allowed, it is not an easy task any more. Note that to deal with the problem of broadcast cover with free riders we must find a solution to the key distribution problem first and fix it. In this thesis, we choose the key distribution of SD scheme, because it has a decent key storage cost and it has the best transmission overhead cost found so far. We will explain the SD scheme in detail in the next section, however, to see the progress in this topic and appreciate SD, we must first consider some older ideas. We first explain three extremely simple broadcast encryption schemes.

- **Key per receiver:** Give every receiver a unique key. To make a broadcast, encrypt with only the keys of the privileged users. This scheme has a

perfect solution to the key distribution problem with only 1 key storage per receiver, however, the center has to make lots of encryptions for each broadcast. Assuming there may be a million privileged users in a broadcast for a cable TV program, it has an awful transmission overhead.

- **Key per subset:** Give all possible subsets of receivers a unique key, and upon a broadcast need, just use the key of the subset that contains only the privileged users. This scheme has a perfect transmission overhead with only 1 encryption, however, every receiver have to store 2^{N-1} keys, which is simply impossible even with a receiver population of hundreds, not thousands or millions.
- **Broadcast to all:** Give the same key to every receiver and encrypt with it in every broadcast. This scheme has an optimum key storage and transmission overhead complexity, however, the revoked users are always free riders, which is contrary to the essence of broadcast encryption. Because even if we want to allow a number of free riders to reduce transmission cost, we have to have a bound on the number of free riders.

To supply the demand for a broadcast encryption scheme with the capability of extracting any possible subsets and requiring decent amount of key storage at the receivers, the tree structure of the LKH scheme was appropriate. In this design, key distribution is done as follows. First consider a binary tree whose leaves are associated with a different receiver. Now suppose we associate every node with a unique key. Then, for every node x , we give the key associated with x to all receivers associated with a leaf which is a descendent of x . If we look from the other way around, to each receiver, we give the keys of the nodes on the path from the root to the leaf associated with that receiver. Since the length of this path is $\log N$ for a balanced tree, key storage complexity turns out to be $O(\log N)$, which is quite good, because even with a million receivers we have approximately twenty keys per receiver.

2.2.1 Tree Scheme and f-Cover Algorithm

Now we explain Tree Scheme of [4], which introduced the idea of allowing free riders in broadcast encryption. The key distribution of the scheme is same as LKH, explained above. In the broadcast cover algorithm, the center first determines a constant, which the authors call *redundancy*, denoted by f , indicating the allowed ratio between the number of receivers that will be able to decrypt and the number of privileged users. For a node, if the ratio of receivers who can decrypt to the privileged receivers under it is less than f , they call it an *f-redundant* node. The F-COVER algorithm starts from the root and traverses the tree in a depth-first or breadth-first order, and checks every node whether it is f-redundant or not. When it encounters with an f-redundant node, it marks that node and does not traverse its descendants any more. At the end, the keys of the marked nodes are the ones that will be used. Thus, transmission overhead is the number of marked nodes. Note that the assignment of the free riders is not optimal in this method. Optimal solution to this problem is given in [19].

2.2.2 The Complete Subtree Scheme

After [4], research on the free rider problem slowed down for a while. The reason was that researchers focused on the special case where there are a small number of revoked users and the real aim is to revoke the illegitimate users. Obviously, free rider approach was not reasonable in this case. Therefore, [7] proposed the complete subtree scheme, which is almost same as the special case of the tree scheme where redundancy is 0. The result of the CS scheme is the same as the tree scheme with no redundancy but its processing method is different. CS scheme processes as follows.

First Steiner tree induced by the leaves associated with the revoked users, $ST(R)$ is found. Then traversing $ST(R)$, every node that hang off $ST(R)$ is marked. At the end, keys of the marked nodes are used to encrypt the broadcast.

Note that the transmission cost can be approximated to $O(r \log N)$ by considering there are r receivers therefore r paths between the corresponding leaves and the root, and these paths are of length $\log N$. However, this is only an upper bound. Considering also the intersections of these paths, they proved that the transmission cost of the CS scheme is $O(r \log N/r)$.

Complete subtree scheme was the last point in broadcast encryption research before SD scheme. However it did not consider the case where free riders are allowed. The solution to this case is given recently by [19] using a dynamic programming approach.

2.2.3 CS with Optimal Free Rider Assignment

Recently, Ramzan and Woodruff [19] proposed an algorithm which assigns free riders optimally in CS scheme. The algorithm does the following.

First it finds the Steiner tree $ST(R)$ of revoked users. Then traverses this tree in postorder by recording the best subsolutions for each node and for each number of free riders allowed. In each calculation of a optimal solution, it uses the optimal subsolutions of the descendant nodes, which was previously solved and recorded. Using this dynamic programming approach, it finds the optimal solution to the problem in $O(rf)$ time.

The authors said that this approach may be applicable to other subset cover schemes like SD, however, they did not give a solution for SD. The problem we attack in this thesis is to find a way to optimally assign free riders for SD scheme.

2.3 The Subset Difference Scheme

We now explain the SD scheme, which is another subset cover scheme, in detail. Note that for a broadcast encryption scheme in subset cover framework, the more the subsets, the easier to cover a privileged receiver set with more accuracy. Using

this idea, Naor et al. [7] proposed the SD scheme which increases the number of subsets by simply assigning more keys to each receiver, but in a clever way.

Key Distribution. As in any subset cover broadcast encryption scheme, a collection of subsets is defined first.

$$\mathcal{S} = \{S_{i,j} | x_j \text{ is a descendant of } x_i\}$$

where

$$S_{i,j} = \{x | x \text{ is a leaf and } x \text{ is a descendant of } x_i \text{ and } x \text{ is not a descendant of } x_j\}$$

Note that the subset $S_{i,j}$ contains the receivers whose associated leaves are descendants of x_i but not descendants of x_j . Intuitively, subsets of SD scheme are exactly the differences of those of CS scheme. Note that a subset of CS scheme rooted by node x , can be found in the subsets of SD which is $S_{parent(x), sibling(x)}$. However, since the root has no parents, full tree cannot be found in the SD subsets, therefore a special subset which is the full tree is added. Each subset $S_{i,j}$ is associated with a key $K_{i,j}$. The key distribution in [7] is performed in a clever way for reducing key storage to $O(\log^2 N)$. A user is not given all of the keys of the subsets which contains it, but given some of them such that it can calculate all of them. This key assignment method has nothing to do with the problem of this thesis, so we skip the details.

Covering Algorithm. For a revoked user set R , First the Steiner tree $ST(R)$ is found. Then a temporary tree T is maintained and initialized to $ST(R)$. Then, the following steps are run until there is only one node in T .

- If there is only one leaf y , add $S_{root,y}$ to the cover, and remove all descendants of the root.
- Else, find two leaves y and z whose lowest-common-ancestor x has no other leaves in T . W.l.o.g. suppose y is on the left of z .
 - If $left(x) \neq y$, add $S_{left(x),y}$ to the cover.

$left(x)$	immediate left child of x
$right(x)$	immediate right child of x
$T(x)$	Subtree rooted at node x
$L(x)$	Subtree rooted at $left(x)$
$R(x)$	Subtree rooted at $right(x)$
$d(x)$	depth of x , that is the distance between the root and x
$r(x)$	number of revoked receivers under x
$p(x)$	number of privileged receivers under x

Table 2.1: Preliminary Notation

- If $right(x) \neq z$, add $S_{right(x),z}$ to the cover.
- Remove all descendants of x from T and make x a leaf node.

Decryption Step. Same as all other broadcast encryption schemes in subset cover framework.

With the covering algorithm above, it is guaranteed that the cover size is at most $2r - 1$. For the formal proof, see [7].

2.4 Definitions

In this section we give the necessary definitions for the algorithms in chapters 3 and 4. First, we recall the notations. We denote the set of all users by U , with $|U| = N$, the set of revoked users for a particular broadcast by R , with $|R| = r$, the set of privileges users by P , with $|P| = p$, and the set of free riders by F , with $|F| = f$.

Throughout the thesis, for a binary tree, we use the preliminary notation in Table 2.1. We give numbers to the nodes in level order, that is we give 1 to the root, and recursively for any node x with number i , we give $2i$ to $left(x)$ and $2i + 1$ to $right(x)$. See Figure 2.3. We denote a node by x_i if it is numbered with i . We denote the subsets in SD scheme in $S_{i,j}$ or S_{x_i,x_j} format interchangeably.

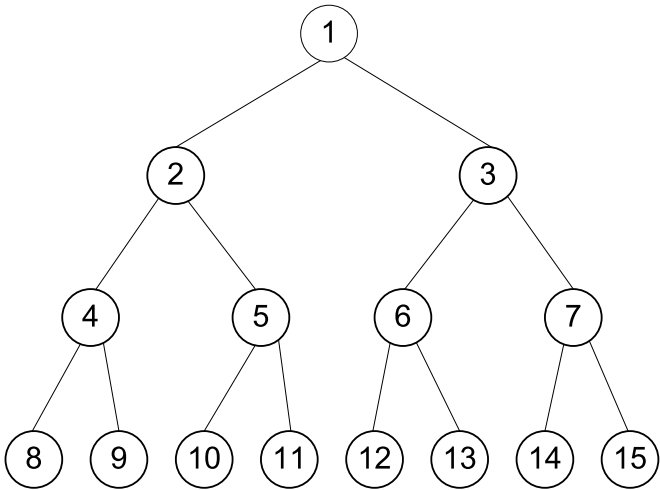


Figure 2.3: Level ordering.

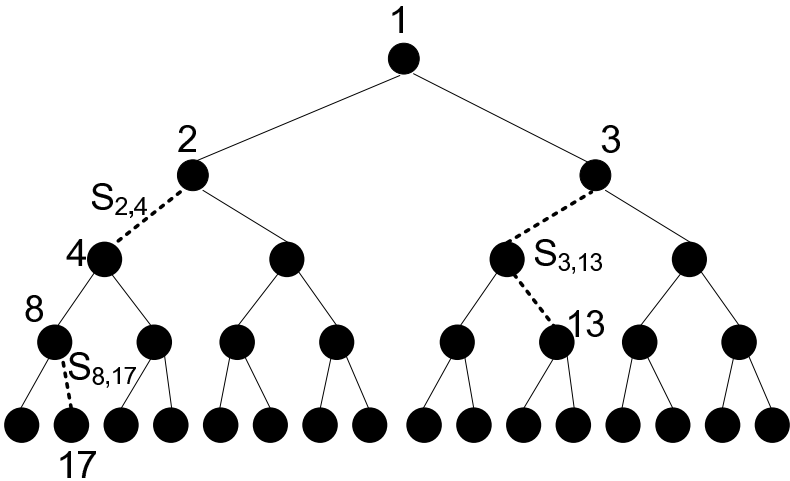


Figure 2.4: A simple SD configuration.

Definition 1 (SD configuration) We call the combination of a tree $\mathcal{T}$ and a collection $\mathcal{C}$ of $S_{i,j}$ subsets on it an SD configuration. See Figure 2.4 This configuration can be a solution to an SD problem or not. But every SD solution is an SD configuration. We also use this term for subtrees specifically as the SD configuration of $T(x)$.

Definition 2 (i-point, j-point) In a particular SD configuration, let $S_{i,j}$ be the subset for covering under node x_i but not node x_j . Then, we call x_i an *i-point* and x_j a *j-point*.

Definition 3 (SD solution) We call an SD configuration an SD solution if it constitutes a valid solution for the current SD problem. i.e. If the union of the $S_{i,j}$ subsets in the collection $\mathcal{C}$ of the configuration covers the privileged users and covers no more than allowed number of free riders.

Definition 4 (Cost) Cost of an SD configuration or an SD solution is defined as the number of $S_{i,j}$ subsets it contains. We also refer to the cost of a SD configuration of $T(x)$ with ‘the cost under x ’.

Note that in exact schemes cost is a function of only revoked leaf positions, but when we have free riders, number of free riders available will also affect the cost. Therefore, minimum cost of an SD configuration becomes a function of revoked user positions and number of free riders available for x . We have slightly different definitions for the optimal solution and its cost, so we postpone their definitions to the related chapters.

For running time efficiency concerns, our algorithms will run only on a special set of nodes, the *meeting points*.

Definition 5 (Meeting Point) We call a node x a meeting point if both $L(x)$ and $R(x)$ contain some revoked users or if it is a leaf node. We call the former type *inner meeting point*.

Since for any inner meeting point x there will be some meeting points in both $L(x)$ and $R(x)$. We denote the highest such meeting points in the left and right subtrees as $mpl(x)$ and $mpr(x)$, respectively.

Chapter 3

Greedy Algorithms

In this chapter we present three greedy algorithms which attacks the problem of efficiently assigning a designated number of free riders to a broadcast encryption instance to reduce transmission overhead.

3.1 Top-Down Algorithm

The aim in this algorithm is to cover the privileged users with as few as possible subsets. Therefore, it is better to have our $S_{i,j}$ subsets such that i is as high as possible. So, we trace the tree from the top to the bottom by checking each node, assuming it is an i-point, whether it satisfies the redundancy bound with some appropriate j-point or not. Now we give the algorithm and then explain the details.

TOP-DOWN-ASSIGN($\mathcal{T}, R, f$)

$f_{ratio} \leftarrow (f + p)/p$

FIND-BEST-EXCLUSION-POINT($root$)

TOP-DOWN-COVER($root$)

FIND-BEST-EXCLUSION-POINT(x)

```

if  $r(x) > 0$ 
  then if  $x$  is a leaf or  $p(x) = 0$ 
    then  $bep(x) \leftarrow x$ 
    return  $x$ 
  else  $y \leftarrow \text{FIND-BEST-EXCLUSION-POINT}(\text{left}(x))$ 
     $z \leftarrow \text{FIND-BEST-EXCLUSION-POINT}(\text{right}(x))$ 
    if  $z = \text{null}$  or  $r(y) > r(z)$ 
      then  $bep(x) \leftarrow y$ 
      return  $y$ 
    else  $bep(x) \leftarrow z$ 
    return  $z$ 
else return null

```

TOP-DOWN-COVER(x)

```

if  $(p(x) + r(x) - p(bep(x)) - r(bep(x))) / (p(x) - p(bep(x))) \leq \text{fratio}$ 
  then  $C \leftarrow C \cup \{S_{x,bep(x)}\}$ 
else if  $r(x) = 0$ 
  then  $C \leftarrow C \cup \{S_{parent(x),sibling(x)}\}$ 
else TOP-DOWN-COVER( $\text{left}(x)$ )
  TOP-DOWN-COVER( $\text{right}(x)$ )

```

The algorithm uses a greedy approach and basically does the following: It first finds the ratio

$$\text{fratio} = (f + p)/p$$

which is called the redundancy of the broadcast.

First step of the algorithm finds a best exclusion point for every node in the tree. Best exclusion point of node x is the descendant of x which has the most number of revoked users and has no privileged users under it. Best exclusion

point is denoted by $bep(x)$. The reason why we choose this node is clear: The minimum redundancy is achieved with such a j-point because we exclude revoked positions as much as possible. This step is simply done recursively in a depth-first manner in $O(N)$ time.

The TOP-DOWN-COVER procedure traces the tree in depth-first order by checking each node x whether $S_{x,bep(x)}$ satisfies the redundancy bound. If it does, the algorithm adds it to the cover and no longer traces its descendants. Then it checks whether all of its receivers are privileged or not. If this is true, it adds $S_{parent(x),sibling(x)}$ to the cover. Otherwise, it recursively traces its children. Eventually, the algorithm finds a cover which satisfies the redundancy bound. This can be proved easily by noting that all $S_{i,j}$ subsets satisfies the redundancy bound, *fratio*. Therefore, the union of these sets should give a redundancy lower than *fratio* as well.

Note that this algorithm requires the redundancy bound to hold at each $S_{i,j}$ taken. However, in an optimal solution it may be the case that a few subsets have a slightly higher redundancy but all other subsets has lower redundancies so that the overall redundancy satisfies the bound. Another restriction is that it does not permit any subsets in the cover to be one under another unless the higher one has no redundancy, i.e. if $S_{i,j}$ is covered, we cannot add another subset $S_{i',j'}$ where i' is a descendant of j unless $S_{i,j}$ covers only original privileged users. However, an optimal solution may have such a configuration. Also note that some free rider quota may remain after this algorithm, which means there is still room for improvement. Next, we will see an algorithm which does not have these drawbacks.

3.2 Bottom-Up Algorithm

In this section, we will describe an algorithm which starts with an initial SD solution, and by making free rider assignments step by step, produces a better SD solution. In each step it assigns free riders to a node, i.e. to all revoked

receivers under a node. The intuition behind the algorithm can be explained with a few observations concerning the assignment of free riders:

- Given an initial SD solution, the aim of the algorithm is to get rid of some $S_{i,j}$ subsets by saturating $T(j)$. So, in an atomic free rider assignment event we assign free riders to the j-point of a subset $S_{i,j}$ in the current configuration and remove this subset. Removal of a subset may or may not reduce the cover size, i.e. transmission overhead.
- The algorithm starts with the exact SD cover and checks each subset $S_{i,j}$ in it to see how much benefit can be obtained by removing it. Once it finds the most beneficial subset, it removes it by saturating $T(j)$ with free riders. When a subset $S_{i,j}$ is removed from a cover, since $T(x_i)$ must be covered, another subset, $S_{parent(i),sibling(i)}$ should be added. See Figure 3.1.

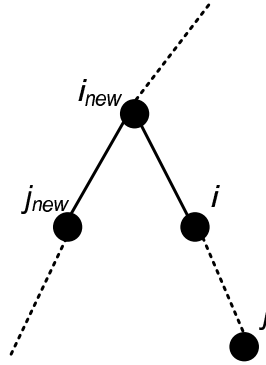


Figure 3.1: Removal of $S_{i,j}$ leads to insertion of $S_{i_{new},j_{new}}$

- When we add a subset to the cover, we need to check whether it can be merged with another subset. This can only happen when i-point of a subset is the j-point of another.

Now we define the profit gained by removing a subset $S_{i,j}$ formally. Consider an initial SD configuration $\mathcal{C}$. When we remove a subset $S_{i,j}$ by saturating its j-point, let the new configuration be $\mathcal{C}'$. The difference $Cost(\mathcal{C}) - Cost(\mathcal{C}')$ is called the *profit* of the subset and denoted by $profit(S_{i,j})$.

The profit that will be obtained by removing an $S_{i,j}$ subset by saturating $T(j)$ can be one of the three:

- **0:** There will be no profit if the subset $S_{parent(i),sibling(i)}$ cannot be merged with any other subset. This happens when neither $parent(i)$ is j-point of another subset nor $sibling(i)$ is an i-point of another subset. See Figure 3.2

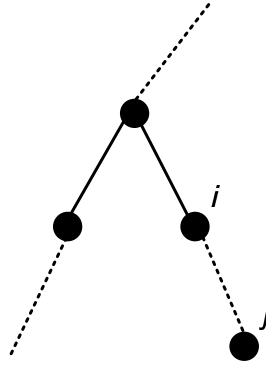


Figure 3.2: No profit

- **1:** A profit of 1 will be gained when the subset $S_{parent(i),sibling(i)}$ can only be merged with one of $S_{i',parent(i)}$ and $S_{sibling(i),j''}$. See Figure 3.3.

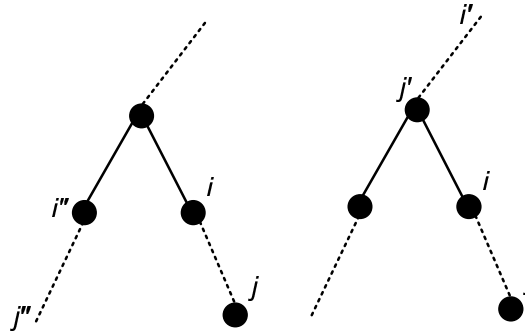


Figure 3.3: Profit of 1

- **2:** As the best case, a profit of 2 will be gained when $S_{parent(i),sibling(i)}$ can be merged with both $S_{i',parent(i)}$ and $S_{sibling(i),j''}$. See Figure 3.4

Now we give the algorithm.

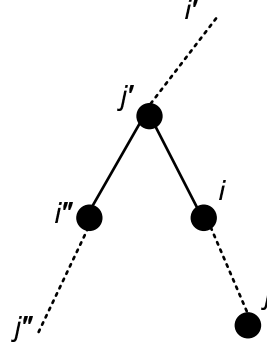


Figure 3.4: Profit of 2

BOTTOM-UP-ASSIGN($\mathcal{T}, R, f$)
 $\mathcal{S} \leftarrow \text{SD-EXACT-COVER}(\mathcal{T}, R)$
 $SL \leftarrow \text{GET-SORTED-SUBSET-LIST}(\mathcal{S}, f)$
 $\mathcal{S} \leftarrow \text{REMOVE-MOST-PROFITABLE-SETS}(\mathcal{S}, SL, f)$

Here, GET-SORTED-SUBSET-LIST algorithm produces a list SL of $S_{i,j}$ subsets, where $r(j) \leq f$, in descending order according to their advantage ratios, which is defined as:

$$ratio_{adv}(S_{i,j}) = \frac{profit(S_{i,j})}{r(j)}$$

REMOVE-MOST-PROFITABLE-SETS($\mathcal{S}, SL, f$)
while $SL \neq \emptyset$
 do $S_{i,j} \leftarrow \text{EXTRACT-FIRST}(SL)$
 $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S_{i,j}\}$
 REMOVE($SL, S_{i,j}$)
 if $J_{parent(i)} = 1$
 then $i_{new} \leftarrow iptr_{parent(i)}$
 $\mathcal{S} \leftarrow \mathcal{S} \setminus S_{i_{new}, parent(i)}$
 REMOVE($SL, S_{i_{new}, parent(i)}$)
 else $i_{new} \leftarrow parent(i)$
 if $I_{sibling(i)} = 1$
 then $j_{new} \leftarrow jptr_{sibling(i)}$
 $\mathcal{S} \leftarrow \mathcal{S} \setminus S_{sibling(i), j_{new}}$

```

        REMOVE( $SL, S_{sibling(i), j_{new}}$ )
    else  $j_{new} \leftarrow sibling(i)$ 
 $f \leftarrow f - r(j)$ 
    CLEAR( $SL, r(j)$ )
    SATURATE( $j$ )
 $\mathcal{S} \leftarrow \mathcal{S} \cup S_{i_{new}, j_{new}}$ 
    INSERT( $SL, S_{i_{new}, j_{new}}$ )

```

REMOVE-MOST-PROFITABLE-SETS($\mathcal{S}, SL, f$) algorithm removes the $S_{i,j}$ sets one by one starting from the most profitable one until the remaining number of free rider assignments is not enough.

First, it gets the most profitable subset $S_{i,j}$ with EXTRACT-FIRST procedure. Then it removes it from the cover and the subset list SL . Before adding the subset $S_{i_{new}, j_{new}}$, it first checks whether it can be merged with other subsets or not. Since there are two possible such subsets, it checks them and removes them if they exist, and updates i_{new} and j_{new} points if necessary. Then it decreases the free rider quota f by $r(j)$ and calls the CLEAR subroutine which clears the list SL from the $S_{i,j}$ subsets having $r(j) > f$. At this point, the procedure have to correct $r(x)$ for each node x above j . This will also yield some changes in the advantage ratios, possibly changing the order of some subsets in SL . This is done in the procedure SATURATE. Finally it inserts the new subset $S_{i_{new}, j_{new}}$.

Now we discuss the running time of this algorithm. Since the dominating procedure is REMOVE-MOST-PROFITABLE-SETS we can omit the running times first two procedures. Note that if $r(j) \leq f$ for all subsets. So, there can be subsets of $f - 1$ different $r(j)$ values, namely 1 to f . Thus, in each step either f decreases by one or all subsets $S_{i,j}$ with a particular $r(j)$ value is removed. Therefore while loop makes at most $O(f)$ iterations. In each iteration, REMOVE, INSERT, and CLEAR procedures has $O(r)$, the SATURATE procedure has $O(\log N \log f)$ complexity. Hence, the overall complexity turns out to be $O(fr + f \log N \log f)$.

Bottom-Up algorithm has the obvious drawback of any greedy algorithm: It does the right thing locally, but this may not turn out to be the best thing globally.

Especially when a large number of free riders are allowed, a worse performance should be expected from this algorithm. On the other hand, it is slower than the Top-down algorithm of the previous section.

3.3 Hybrid Algorithm

So far, we have described two greedy algorithms with their respective advantages and disadvantages. Below, we describe an algorithm where we use the Top-down algorithm to get the initial subset collection for the Bottom-up algorithm instead of using the exact cover solution. This yields a result faster but a little worse transmission overhead compared to the original Bottom-up algorithm. Here is the algorithm:

```

HYBRID-ASSIGN( $\mathcal{T}, R, f$ )
 $\mathcal{S} \leftarrow \text{TOP-DOWN-ASSIGN}(\mathcal{T}, R, f)$ 
 $SL \leftarrow \text{GET-SORTED-SUBSET-LIST}(\mathcal{S}, f)$ 
 $\mathcal{S} \leftarrow \text{REMOVE-MOST-PROFITABLE-SETS}(\mathcal{S}, SL, f)$ 

```

The algorithm first runs the TOP-DOWN-ASSIGN subroutine, then continues same as the Bottom-up algorithm. Forms the sorted subset list according to the averages and remove one by one at each step until spending all free rider quota.

Note that the running time of the algorithm is somewhere between that of the Top-down algorithm and the Bottom-up algorithm, because it basically runs the Top-down algorithm and then calls some other procedures, or looking from the other way around, it arrives at the second line of the Bottom-Up algorithm with a smaller f value yielding a much faster run for the last two lines.

In Chapter 5, we will compare the performance of this algorithm with the previous two.

Chapter 4

Optimal Free Rider Assignment for the SD Scheme

In this chapter, we give two optimal free rider assignment algorithms for the SD scheme. Both algorithms are based on a dynamic programming approach. In fact, the algorithms are similar in essence but the difference is that the first algorithm is more intuitive but the second one is faster.

4.1 The Optimization Algorithms

We will describe an algorithm based on a dynamic programming approach in order to get the optimal SD solution for given a set of revoked users R , and a number of free riders f . We will run the algorithm on meeting points and calculate the cost, with all possible assignments of free riders, using the costs of the closest descendant meeting points from left and right. Now, as the first step of dynamic programming, we characterize the structure of an optimal solution.

Definition 6 (Cost($\mathbf{x}, n$)) *We denote the cost of an SD configuration for $T(x)$ by assigning n free riders, as $Cost(x, n)$. We define this cost as the minimum possible number of $S_{i,j}$ subsets covering leaves in $T(x)$.*

Consider a node x , let $y = mpl(x)$ and $z = mpr(x)$. Suppose f free riders are available to assign under x . Then, if we assign $f(y)$ of these free riders under y and $f(z)$ of them under z , the overall cost of x becomes:

$$Cost(x, f(x)) = Cost(y, f(y)) + Cost(z, f(z)) + E_y + E_z$$

Here, E_y and E_z denotes the possible costs on the paths x through y and z respectively.

Hence, in an optimal solution the summation of the costs of these four factors should be minimized. We can come up with a recursive solution as follows.

$$Cost(x, f(x)) = \min_{\substack{0 \leq f(y) \\ 0 \leq f(z) \\ f(y)+f(z)=f(x)}} \{Cost(y, f(y)) + Cost(z, f(z)) + E_y + E_z\}, \quad (4.1)$$

where $y = mpl(x)$, $z = mpr(x)$, and E_y and E_z are the possible extra costs originating from the paths from x to y and z respectively. Each of the last two factors can bring at most one extra cost, and it depends on some other conditions. We will explain these conditions in next two sections.

4.1.1 The Basic Algorithm

In this section, we first give the essence of the algorithm, then explain the details of the dynamic programming formulation, and then we give the conditions that make $E_y = 1$ and $E_z = 1$.

In the recursive step of the algorithm, which was formulated in (4.1), we derive the cost for a node using the costs at the lower meeting points and the paths towards them. Apparently, any node may be assigned any number of free riders in an optimal solution, therefore we should calculate a cost for all nodes and for all possible number of free riders that can be assigned to them.

In the basic algorithm, with $Cost(x, n)$, we denote the minimum number of $S_{i,j}$ subsets under node x , by assigning *exactly* n free riders under node x . Note that the optimal solution to a problem with f free riders available, may be obtained by using less than f free riders. To visualize this, suppose a receiver setting, where all users are privileged except the ones under a particular node which is not the root. Without assigning free riders, we have a cost of 1. However, when we assign a free rider, the cost increases. So, we cannot guarantee that the more free riders we assign, the less the cost is. As a result, we have to try assigning from 0 to f free riders to the root.

We can now explain how to calculate E_y and E_z . Due to their symmetry, w.l.o.g., we will only explain for E_y . It syntactically stands for “extra cost between x and y ”. Semantically, it is the cost possibly originating from the path through x to y . There are three possibilities.

- Either there is no need for an $S_{i,j}$ subset lying between x and y ,
- or there must be an $S_{i,j}$ subset but it can be merged with the $S_{i,j}$ subset right underneath y ,
- or there must be an $S_{i,j}$ subset and it cannot be merged with another one.

Let us investigate when one would need an $S_{i,j}$ subset between x and y .

- $r(y) > f(y)$. First, number of revoked users under y must be greater than $f(y)$. Otherwise, that is if $r(y) = f(y)$, we would not need any subsets because we have enough free riders to fully saturate and make all $r(y)$ revoked users under y free riders. Then, we leave the covering of the users under y to a subset of higher levels. There will be a subset in the higher levels of the tree, whose i-point is an ancestor of y , but its j-point is not. Thus, to get a 1 for E_y , it must be the case that $r(y) > f(y)$. This condition is necessary but not sufficient.
- $y \notin I$. Another necessity is that y is not an i-point. Because, even if there exists an $S_{i,j}$ subset between x and y , it would be merged with the subset

underneath y . So, it does not matter whether there will be a subset between x and y because it will cancel anyway and bring no cost. This condition is also necessary but not sufficient.

- Given the above two conditions are both true, at least one of the following conditions must also hold in order to guarantee $E_y = 1$.

- $d(y) - d(x) \geq 2$. If there are some nodes between x and y , that is $d(y) - d(x) \geq 2$, there will be an extra subset between x and y , which will make $E_y = 1$ certainly, because of two reasons. First, since there are some nodes which are not meeting points, between x and y , one children of each of these nodes has privileged users under them, therefore there has to be an $S_{i,j}$ subset such that its i-point is above these nodes. So, there are only two possibilities: x , and the left child of x . Second, since y has both privileged and remaining revoked users, there must be a j-point somewhere under y . But, since y is not an i-point, this j-point cannot be the j-point of x or $left(x)$, one of which we have proved to be an i-point. Then the only possibility for the j-point is y . Thus, we end up with an $S_{i,j}$ subset, whose i-point is x or $left(x)$, and j-point is y . Therefore, in both cases, we will have an extra cost, that is $E_y = 1$.

- $r(z) = f(z)$. Now, assume the condition above does not hold, meaning y is the immediate left child of x . Then only possibility for an $S_{i,j}$ subset between x and y is the subset whose i-point is x and j-point is y . Note that this means the subtree of the right child of x is fully saturated. Therefore z must be fully saturated. Then we must have $r(z) = f(z)$ to saturate z and get $E_y = 1$.

Thus, the first two conditions, together with either $d(y) - d(x) \geq 2$ or $r(z) = f(z)$ is sufficient to get an extra subset between x and y and make $E_y = 1$.

Throughout the algorithm, we keep the meeting points in an array called MP , for efficient implementation. MP is prepared by the FIND-MEETING-POINTS

subroutine as follows. First, MP is initialized to leaves that are associated with the revoked users and mark all of them. Then we choose two marked meeting points mp_1 and mp_2 such that their lowest common ancestor mp_c has no other descendant marked meeting points. We add mp_c to MP , mark it, and unmark mp_1 and mp_2 . We progress until only one marked meeting point remains. Eventually, we end up with a meeting point list in which a meeting point always comes before its ancestors. Therefore, when we trace this array in order, we always arrive at a meeting point after its descendants. We also know that first r are the leaves and the rest, $r + 1$ through $2r - 1$ are the inner meeting points.

For any meeting point x and any possible number of free riders assigned to it, we also need to keep track of the optimal cost found for x , number of free riders assigned to left subtree of x , and a flag indicating whether x is an i-point or not in the optimal configuration. The arrays we use to hold this information are C_x , L_x , and I_x , respectively.

FREE-RIDER-ASSIGN-BASIC ($\mathcal{T}, R, f$)

$MP \leftarrow \text{FIND-MEETING-POINTS}(R)$

for $i \leftarrow 1$ **to** r

$x \leftarrow MP[i]$

$C_x \leftarrow [0, 0]$

$I_x \leftarrow [0, 0]$

for $i \leftarrow r + 1$ **to** $2r - 1$

$x \leftarrow MP[i]$

$y \leftarrow mpl(x)$

$z \leftarrow mpr(x)$

for $f(x) \leftarrow 0$ **to** $\min(r(x), f)$

$C_x[f(x)] \leftarrow \infty$

for $f(y) \leftarrow \max(f(x) - r(z), 0)$ **to** $\min(r(y), f(x))$

$f(z) = f(x) - f(y)$

$tempcost \leftarrow Cost(y, f(y)) + Cost(z, f(z)) + E_{y, f(y)} + E_{z, f(z)}$

if $tempcost < C_x[f(x)]$ **or**

$\left(tempcost = C_x[f(x)] \text{ and } (r(y) = f(y) \text{ or } r(z) = f(z)) \right)$


```

then  $C_x[f(x)] \leftarrow tempcost$ 
       $L_x[f(x)] \leftarrow f(y)$ 
      if  $f(y) = r(y)$  or  $f(z) = r(z)$ 
        then  $I_x[f(x)] = 1$ 
        else  $I_x[f(x)] = 0$ 

/* Now calculating the optimal cost for the root */
result  $\leftarrow \infty$ 
for  $f(root) \leftarrow 0$  to  $f$ 
  rootcost  $\leftarrow C_{MP[2r-1]}[f(root)]$ 
  if  $depth(MP[2r-1]) \neq 0$  and  $I_{MP[2r-1]}[f(root)] \neq 1$ 
    then rootcost  $\leftarrow rootcost + 1$ 
  if rootcost  $< result$ 
    then result  $\leftarrow rootcost$ 
return result

```

where

$$E_{y,f(y)} = \begin{cases} 1 & \text{if } r(y) > f(y) \text{ and } y \notin I \text{ and } (d(y) - d(x) \geq 2 \text{ or } r(z) = f(z)) \\ 0 & \text{otherwise} \end{cases}$$

and similarly,

$$E_{z,f(z)} = \begin{cases} 1 & \text{if } r(z) > f(z) \text{ and } z \notin I \text{ and } (d(z) - d(x) \geq 2 \text{ or } r(y) = f(y)) \\ 0 & \text{otherwise} \end{cases}$$

First, the meeting points are found. This is a rather easy task, which can be performed very fast, starting from revoked users and run in a bottom-up fashion, as we mentioned earlier. Then, we fill C and I arrays of the leaves with zeros. This is trivial since there cannot be any subsets under a leaf and it cannot be an i-point. In the rest of the algorithm, we trace the meeting points in an bottom-up fashion, so that when we arrive any meeting point x , we have both solutions for $mpl(x)$ and $mpr(x)$. At each meeting point x , we find optimal solutions for all possible free rider count for x . For a particular number of free riders $f(x)$ for x , we find all costs and corresponding I and L arrays with possible partitions of free riders assigned to x to its children. Among these, we take the one with

minimum cost. If there occurs a tie between two or more partitions, we prefer the one which makes x an i-point. This is very intuitive because when x is an i-point, there is a probability for it to be merged with the subset just above. Otherwise, it cannot be merged. Thus, making x an i-point is at least as good as not making so. Algorithm finally considers –for all possible number of free riders– the costs of the root meeting point together with the cost over it, since there can possibly be other nodes over the highest meeting point, which means it is not the root of the real tree. This can occur when all revoked users are grouped under a node which is not the root. In such a situation, we must add a cost of 1 for the $S_{i,j}$ subset, whose i-point is the root of the real tree, and j-point is the root of the Steiner tree, unless the root meeting point is an i-point. The running time of the algorithm is $O(rf^2)$ in the worst case, yet this is a loose bound.

4.1.2 Improved Algorithm

In this section, we present a slightly improved algorithm. Basically it has the same essence as the basic algorithm but by changing the definition of $Cost(x, n)$ we save the computation for finding actually how many free riders leads to the minimum cost which is done at the root meeting point.

In this algorithm, with $Cost(x, n)$, we denote the minimum cost under node x , by assigning *at most* n free riders under node x .

The basic algorithm and this one differ in two points. First, we do not calculate the cost of the root meeting point with free rider counts less than f . So, instead of taking minimum among all free rider counts, we take $Cost(root, f)$ as the result directly. Second, E_y and E_z are defined differently in this algorithm. In fact, we introduce a few more cases in their definitions. We will now explain these differences in detail.

The first difference is more intuitive and clear. With the new $Cost(x, n)$ definition, we handle all free rider counts equal to and less than f at once. To make it even more clear, we do not need to calculate $Cost(x, n')$ where $n' < f$

because we are sure that there are f free riders available for the root. So, there is no need to bound the number of free riders to any number less than f and calculate a cost for the root meeting point.

The difference we made in the definition of $Cost(x, n)$ does not change the structure of an optimal sub-solution at the high level description. We still have the recursive formula (4.1). However, we need to change the definitions of the extra costs E_y and E_z .

We now explain how to calculate E_y and E_z . As we did in the basic algorithm, due to their symmetry, w.l.o.g., we will only explain for E_y . We have the same three possibilities as in the basic algorithm.

- Either there is no need for an $S_{i,j}$ subset in this path
- or there must be an $S_{i,j}$ subset but it can be merged with the i - j pair just below it.
- or there must be an $S_{i,j}$ subset and it cannot be merged with another one

Let us now investigate when one would need a subset between x and y .

- First, number of revoked users under y must be greater than $f(y)$. Otherwise, $r(y) = f(y)$, and this means we fully saturate and make all $r(y)$ revoked users under y free riders. Then, we leave the covering of the users under y to a subset of higher levels. Thus, to get a 1 for E_y , it must be the case that $r(y) > f(y)$. Notice this condition is same as the first condition in the basic algorithm. Again, it is necessary but not sufficient.
- Second, y must not an i -point in this E_y definition as well. Because, again similar to the E_y definition of the previous algorithm, even if there exists a subset between x and y , it would be merged with the subset underneath y . This condition is also necessary but not sufficient.
- Given the above two conditions are both true, at least one of the following conditions must also hold in order to guarantee $E_y = 1$.

- There must be some nodes between x and y . Then, there will be an extra subset between x and y , which will make $E_y = 1$ certainly, because of two reasons which are exactly same as the ones in E_y definition of the previous algorithm.
- Now, assume the condition above does not hold, meaning y is the immediate left child of x . Then only possibility for a subset $S_{i,j}$ between x and y is the subset whose i-point is x and j-point is y . Note that this means the subtree of the right child of x is fully saturated. Therefore, z must be fully saturated. Then there are two necessary conditions which are sufficient together to saturate z and get $E_y = 1$.
 - * First, we must have enough free riders to be able to saturate z , that is $r(z) \leq f(z)$. This condition is obviously necessary but not sufficient alone. We need the following condition as well.
 - * Note that when we have an $S_{i,j}$ subset with x as the i-point and y as the j-point, we fully saturate the subtree rooted at the right child of x , which is either z or the highest node between x and z . However, in order for this subset to make sense, there must be some privileged users under this saturated subtree. That is, it should not be the case that this subtree is fully revoked and we add a subset with a possible extra cost, just because we are able to. Therefore, it must be the case that $p(z) > 0$ or $d(z) - d(x) \geq 2$.

If these two conditions both hold, then one should fully saturate z . Otherwise it is not advantageous.

Thus, the first two conditions, together with either $d(y) - d(x) \geq 2$ or $(r(z) \leq f(z) \text{ and } (p(z) > 0 \text{ or } d(z) - d(x) \geq 2))$. is sufficient to get an extra cost between x and y .

We still have the same data structures for a node x : C_x , I_x , and L_x . This time, i^{th} element of C_x holds the optimal cost found for $T(x)$ by assigning *at most* i free riders under x . Then, I_x and L_x is defined correspondingly. MP array is exactly same as the previous algorithm.

FREE-RIDER-ASSIGN-IMPROVED ($\mathcal{T}, R, f$)
 $MP \leftarrow \text{FIND-MEETING-POINTS}(R)$
for $i \leftarrow 1$ **to** r
 $x \leftarrow MP[i]$
 $C_x \leftarrow [0, 0]$
 $I_x \leftarrow [0, 0]$
for $i \leftarrow r + 1$ **to** $2r - 1$
 $x \leftarrow MP[i]$
 $y \leftarrow \text{mpl}(x)$
 $z \leftarrow \text{mpr}(x)$
 for $f(x) \leftarrow 0$ **to** $\min(r(x), f)$
 if $i = 2r - 1$
 then $f(x) \leftarrow \min(r(x), f)$
 $C_x[f(x)] \leftarrow \infty$
 for $f(y) \leftarrow \max(f(x) - r(z), 0)$ **to** $\min(r(y), f(x))$
 $f(z) = f(x) - f(y)$
 $\text{tempcost} \leftarrow \text{Cost}(y, f(y)) + \text{Cost}(z, f(z)) + E_{y, f(y)} + E_{z, f(z)}$
 if $\text{tempcost} < C_x[f(x)]$ **or**
 $(\text{tempcost} = C_x[f(x)] \text{ and } (r(y) = f(y) \text{ or } r(z) = f(z)))$
 then $C_x[f(x)] \leftarrow \text{tempcost}$
 $L_x[f(x)] \leftarrow f(y)$
 if $f(y) = r(y)$ **or** $f(z) = r(z)$
 then $I_x[f(x)] = 1$
 else $I_x[f(x)] = 0$
 $\text{result} \leftarrow C_{MP[2r-1]}[f]$
if $\text{depth}(MP[2r - 1]) \neq 0$ **and** $I_{MP[2r-1]}[f] \neq 1$
 then $\text{result} \leftarrow \text{result} + 1$
return result

where

$$E_{y,f(y)} = \begin{cases} 1 & \text{if } r(y) > f(y) \text{ and } y \notin I \text{ and} \\ & \left(d(y)-d(x) \geq 2 \text{ or } \left(r(z) \leq f(z) \text{ and } (p(z) > 0 \text{ or } d(z) - d(x) \geq 2) \right) \right) \\ 0 & \text{otherwise} \end{cases}$$

and similarly,

$$E_{z,f(z)} = \begin{cases} 1 & \text{if } r(z) > f(z) \text{ and } z \notin I \text{ and} \\ & \left(d(z)-d(x) \geq 2 \text{ or } \left(r(y) \leq f(y) \text{ and } (p(y) > 0 \text{ or } d(y) - d(x) \geq 2) \right) \right) \\ 0 & \text{otherwise} \end{cases}$$

In this version of the algorithm, the improvement is on the number of cost calculations. The difference comes from the root node. Although the theoretical bound remains same, the actual cost decreases. This improvement can be observed in the next section, which contains our experimental results.

Chapter 5

Experimental Results

In this chapter, the experiments conducted to compare the performances of the algorithms explained in chapters 3 and 4, are presented. To see the advantage of the free riders, the algorithms are also compared with the original SD scheme of [7], which is the best scheme in terms of transmission cost discovered so far in no-free-rider case.

The organization of the chapter is as follows: We first describe the evaluation parameters. Then, we briefly explain the implementation. Finally, we give the experimental results which compare the performances of our algorithms as well as the original SD scheme, in terms of transmission overhead with different values of the input parameters.

5.1 Evaluation Parameters

As we explained in detail in Chapter 2, we fix the key distribution to the one described in original SD scheme, and focus on the transmission overhead. Therefore, the primary evaluation parameter in this thesis is the transmission overhead, which is a function of the following four variables:

- N , number of all receivers.
- R , the set of revoked receivers.
- r , the number of revoked receivers.
- f , the number of free riders allowed. i.e. free rider quota.

Another performance measure is, like almost all computer-related work, the running time complexities of the algorithms.

5.2 Implementation Issues

Consider the four factors that affect the transmission overhead we mentioned above. Throughout the experiments, we choose and fix a reasonable number $N = 1024$ for the number of receivers, and we take 100 runs for every number of privileged users possible, i.e. from 1 to 1024. We assume that the free rider quota is directly proportional to the number of privileged users. So use the *fratio* definition of :

$$fratio = (f + p)/p$$

Finally, we take the average transmission cost for a particular number of privileged users and corresponding free rider quota. We run this experiment for different values of *fratio*, namely $2/1$, $4/3$, $8/7$, $8/5$, $16/15$, $16/13$. Thus we observe the changes in the performances of the algorithms when a larger free rider quota is available. We show the results on one graph per one *fratio* value.

We also consider the execution times of the experiments explained above. We are particularly interested in the comparison of the two optimal algorithms and we want to see to what extent the greedy algorithms are close to them in terms of computation time.

5.3 Experiments

In this section, we present the results of the experiments with appropriate graphs and tables and discuss them.

5.3.1 Transmission Overhead Experiments

The following graphs shows the transmission costs of the algorithms with respect to the target privileged user set size p and the corresponding free rider quota f , which is calculated as:

$$f = p(fratio - 1)$$

A different graph is presented for each $fratio$ value in Figures 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6.

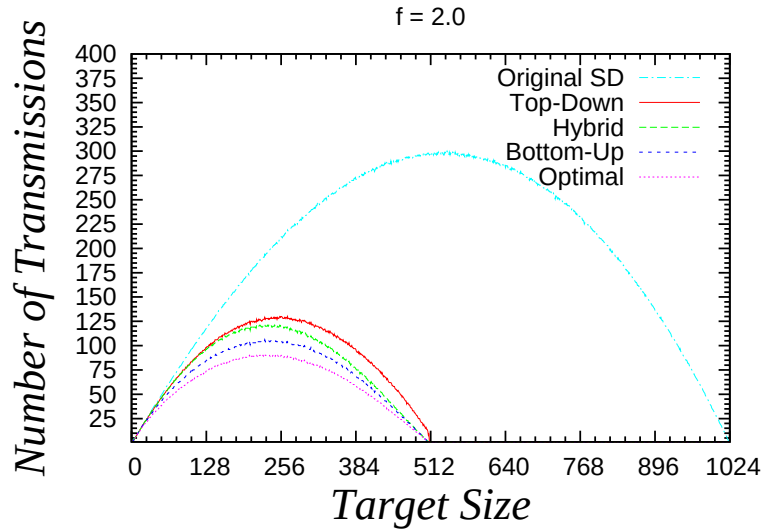


Figure 5.1: Transmission costs of our algorithms with $f = 2.0$.

First of all note that as the free rider quota decreases, all of the algorithms converge to the original SD scheme as we expected. Another point to observe is that the transmission costs of the algorithms with free riders become 1 when

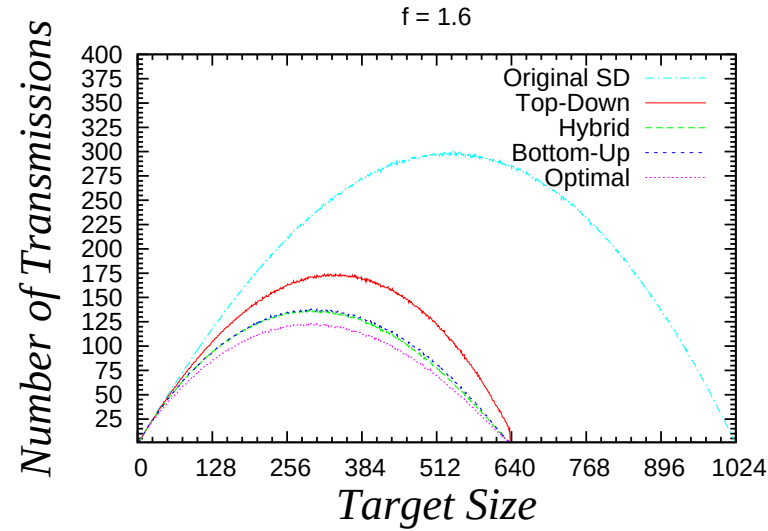


Figure 5.2: Transmission costs of our algorithms with $f = 1.6$.

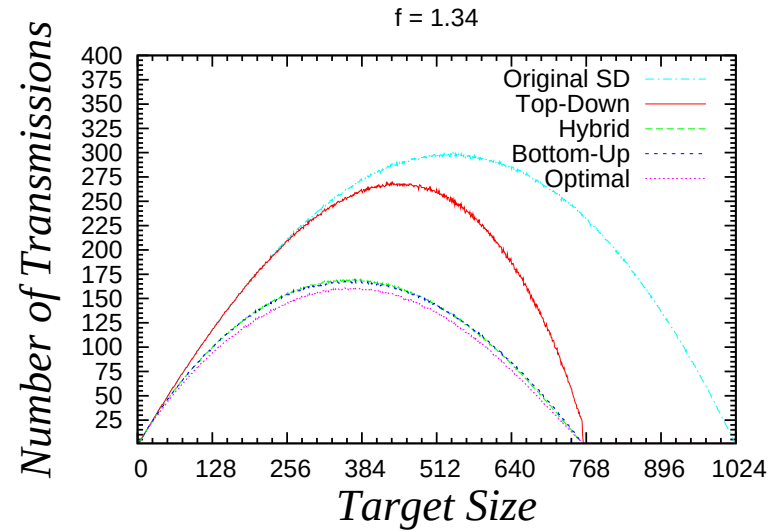


Figure 5.3: Transmission costs of our algorithms with $f = 1.34$.

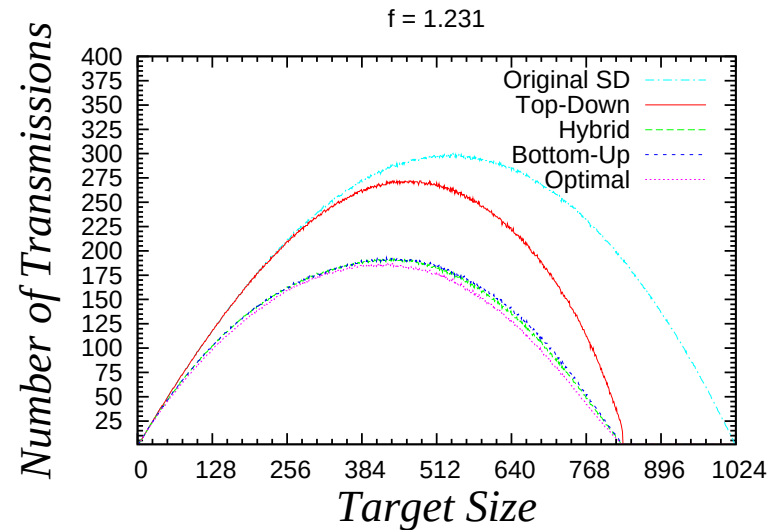


Figure 5.4: Transmission costs of our algorithms with $f = 1.231$.

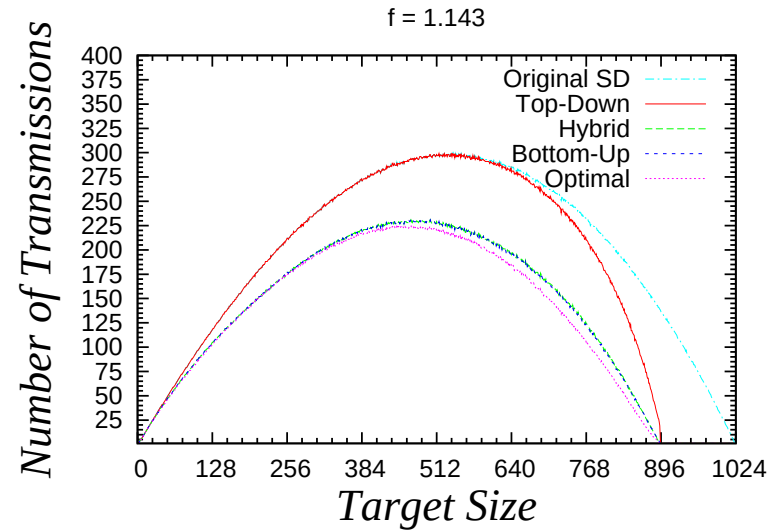


Figure 5.5: Transmission costs of our algorithms with $f = 1.143$.

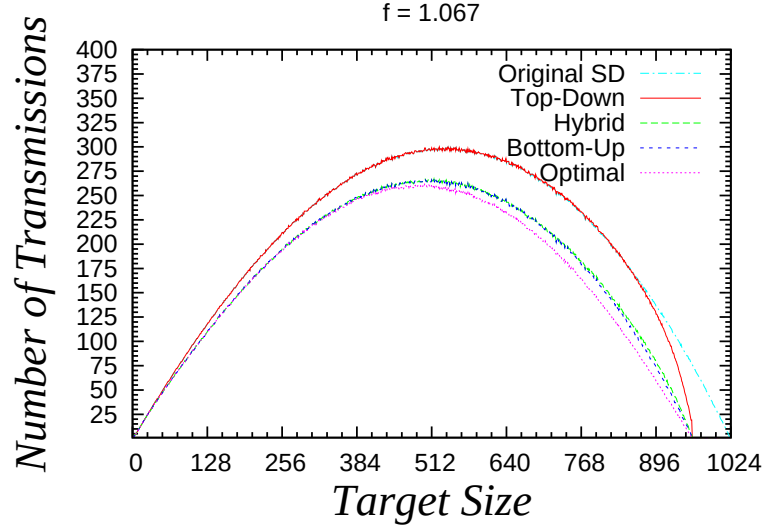


Figure 5.6: Transmission costs of our algorithms with $f = 1.067$.

$$N/p \geq f \text{ratio}$$

because that means $r \leq f$ and one can saturate all receivers with the free rider quota available and make a single broadcast to everyone.

As the free rider quota decreases, the performance of the Top-Down algorithm gets worse faster than the other algorithms. This can be explained by the following strong restriction of the Top-Down algorithm: It operates with the free rider ratio instead of the free rider count, and requires that this ratio holds in every subtree. Therefore, for example, when $f = 1.067 = 16/15$, the Top-down algorithm even has no chance in subtrees having less than 16 receivers, because in an $S_{i,j}$ subset in such a subtree, there can be at most 15 receivers, therefore the only ratio smaller than $16/15$ is 1, which leads to an exact cover which has no free riders. Hence, Top-down algorithm tends to operate in the higher levels of the tree and once it arrives at the lower levels that it cannot make improvements, a portion of the free rider quota remains and cannot be used. Actually this is one of the reasons why we devised the Hybrid algorithm.

The Bottom-up algorithm and the Hybrid algorithm has close performances for lower free rider quota. Actually, the only significant difference between the performances of these algorithms is observed in the case where $f = 2.0$. Note that this case is also the one where the Top-down algorithm works best. There is a strong relation between these two observations: Since the the Hybrid algorithm is actually the successive runs of Top-down and Bottom-up algorithms, when $f = 2.0$ the Top-down algorithm spends the free rider quota much more and leaves less free riders for the Bottom-up algorithm, therefore the Bottom-up algorithm cannot reduce the transmission cost much. However, when the Bottom-up algorithm works alone, it uses the portion of the free rider quota used by the Top-down algorithm within the Hybrid algorithm more effectively and gets a better performance compared to the Hybrid algorithm.

Note that the Bottom-up and Hybrid algorithms has a transmission cost pretty close to that of the optimal algorithms. The reason is that the drawbacks of the Bottom up algorithm is seen in very extreme cases. The advantage of the optimal algorithms compared to the Bottom-up algorithm is that they handle the situations where making a revoked user free rider does not have an advantage immediately but it leads to a better configuration afterwards.

5.3.2 Running Time Comparisons

In this section we compare the running times of:

- The two optimal algorithms
- The greedy algorithms with each other

Table 5.1 shows the execution times of different algorithms with different free rider ratios. For each entry, the algorithms are run 100 times with each possible number of privileged users from 1 to 1024.

The results on Table 5.1 show that the greedy algorithms, especially the Top-down algorithm, run much more faster than the optimal algorithms, as expected.

Algorithm \ f_{ratio}	1.067	1.143	1.231	1.34	1.6	2.0
Top Down	49	48	48	48	48	49
Hybrid	266	419	429	483	259	70
Bottom Up	238	436	622	781	1007	1197
Optimal Basic	5384	5440	5424	5491	5544	5633
Optimal Improved	4142	4167	4188	4232	4283	4307

Table 5.1: Running times of our algorithms in terms of seconds: For each entry on the table, our algorithms are run 100 times for each possible target set size from 1 to 1024 with the corresponding f values.

For small f_{ratio} values, Even the Hybrid and the Bottom-up algorithms runs approximately 15 times faster than the improved optimal algorithm. But as the number of free riders increase, Bottom-up algorithm slows down but it is still 4 times faster than the improved optimal algorithm when $f_{ratio} = 2.0$.

The running time of the Hybrid algorithm first increases with as f_{ratio} increases because for smaller f_{ratio} , the Top-down procedure within the Hybrid algorithm does not operate much and a big portion of the free rider quota remains. However, when f_{ratio} gets larger, the Top-down subroutine in the Hybrid algorithm spends most of the free rider quota and leaves a little work to the Bottom-up subroutine.

As far as the optimal algorithms are concerned the improved optimal algorithm runs approximately 23% faster than the basic version. The main factor making this difference is the extra calculation in the root of the tree for finding the actual free rider count needed to get the minimum cost. Remember that this information was stored inheritedly in the improved version.

Chapter 6

Conclusion and Future Work

In this thesis, we presented three greedy and two optimal algorithms for the problem of placing free riders in SD broadcast encryption scheme which has the lowest transmission cost in the original form. Our first algorithm which uses a greedy top down approach, works extremely fast for any free rider quota, but has the following drawback: It may not use all of the free rider quota, and it may use a large free rider quota fast and carelessly, therefore misses possible configurations which are more efficient. We get rid of the major drawbacks of our first algorithm in the second algorithm using a greedy bottom up approach, which places free rider positions more slowly and carefully and uses all of the free rider quota. However, it still cannot achieve the optimal transmission cost and it gets slower when the free rider quota increases. Noting that the two greedy approaches have different drawbacks we offer an algorithm, which merges them and gets a result faster than the bottom up algorithm and performs better than the top down algorithm. The hybrid algorithm still has transmission costs quite close to the results of the bottom up algorithm. Then we devise an algorithm based on a dynamic programming approach which finds the best free rider placement leading to the minimum transmission overhead. Then we improve this algorithm by slightly changing its optimal substructure, and we end up with a more than 20% faster algorithm.

As a result, for the problem of placing free riders in broadcast encryption

schemes we have the optimal algorithm in terms of transmission cost discovered *so far*. The running time is $O(rf^2)$ in the worst case. The bottom-up algorithm achieves transmission costs quite close to the optimal algorithm in $O(fr + f \log N \log f)$ time. The experiments we conducted shows that it is possible to reduce transmission overhead for the SD scheme by allowing free riders.

The research for better broadcast encryption schemes is going on heavily. Recently, researchers [11, 12, 13, 14, 15, 16, 17, 18] find out other broadcast encryption schemes based on some techniques such as using one-way functions, which satisfies log-key restriction, with good transmission costs. As soon as a better scheme is found, it will be an open problem whether it yields better results when free riders are allowed compared to SD scheme or not. In fact, even if a scheme is not as good as SD in the exact cover case, we cannot guarantee it is not better in free rider case. The subset structure of a scheme may be much more appropriate for allowing free riders and it can turn out to be better in terms of transmission overhead compared to SD with free riders.

The variants of SD, LSD [8] and SSD [10] schemes may be treated by allowing free riders. Actually, the algorithms we presented are applicable to these schemes with a few differences. The reason is the major difference between these two schemes and original SD is the key distribution algorithm. However, the free rider problem can be treated by just considering the broadcasting step. For example by considering the layers in LSD scheme, we may apply our techniques to LSD scheme. The same approach works for SSD as well.

Another observation that can be subject to future work is that when a broadcast encryption system allows free riders the key distribution step can also be relaxed. Because unlike the exact cover case, all possible subsets do not necessarily have to be obtained. Actually it is enough for such a system to be able to find a slightly larger subset which covers the original one. So, it may be possible to use the free riders to reduce the key storage.

Bibliography

- [1] Berkovits, S. How to Broadcast a Secret. *Advances in Cryptology: EUROCRYPT 1991 (LNCS 547)*, pp. 535-541, 1991.
- [2] Fiat, A., Naor, M. Broadcast Encryption. *Advances in Cryptology: CRYPTO 1993 (LNCS 773)*, pp. 480-491, 1993.
- [3] Luby, M., Staddon, J. Combinatorial Bounds for Broadcast Encryption. *Advances in Cryptology: EUROCRYPT 1998 (LNCS 1403)*, pp. 512-526, 1998.
- [4] Abdalla, M., Shavitt, Y., Wool, A. Key Management for Restricted Multicast using Broadcast Encryption. *ACM Trans. on Networking*, vol. 8, no. 4. 2000.
- [5] Wallner, D. M., Harder, E. J., Agee, R. C. Key Management for Multicast: Issues and Architectures. *IETF Network Working Group, Request for Comments:2627*, <ftp://ftp.ietf.org/rfc/rfc2627.txt>, 1999.
- [6] Wong, C. K., Gouda, M., Lam, S. Secure Group Communications Using Key Graphs. *Proceedings of SIGCOMM 1998*, 1998.
- [7] Naor, D., Naor, M., Lotspiech, J. Revocation and Tracing Schemes for Stateless Receivers. *Advances in Cryptology: CRYPTO 2001 (LNCS 2139)*, pp. 41-62, 2001.
- [8] Halevy, D., Shamir, A. The LSD Broadcast Encryption Scheme. *Advances in Cryptology: CRYPTO 2002 (LNCS 2442)*, pp. 47-60, 2002.
- [9] Horwitz, J. A Survey of Broadcast Encryption. *Manuscript*, 2003.

- [10] Goodrich, M. T., Sun, J. Z., Tamassia, R. Efficient Tree Based Revocation in Groups of Low-State Devices. *Advances in Cryptology: CRYPTO 2004 (LNCS 3152)*, pp. 511-527, 2004.
- [11] Asano, T. Reducing Storage at Receivers in SD and LSD Broadcast Encryption Schemes. *WISA 2003(LNCS 2908)*, pp. 317-332, 2004.
- [12] Hwang, J. Y., Lee, D. H., Lim, J. Generic Transformation for Scalable Broadcast Encryption Schemes. *Advances in Cryptology: CRYPTO 2005 (LNCS 3621)*, pp. 276-292, 2005.
- [13] Aravamuthan, S., Lodha, S. An Optimal Subset Cover for Broadcast Encryption. *Advances in Cryptology: INDOCRYPT 2005 (LNCS 3797)*, pp. 221-231, 2005.
- [14] Yoo, E. S., Jho, N.-S., Cheon, J. H., Kim, M.-H. Efficient Broadcast Encryption Using Multiple Interpolation Methods. *Information Security and Cryptology ICISC 2004 (LNCS 3506)*, pp. 87-103, 2005.
- [15] Jho, N.-S., Yoo, E. S., Cheon, J. H., Kim, M.-H. New Broadcast Encryption Scheme Using Tree-Based Circle. *Digital Rights Management Workshop 2005*, pp. 37-44, 2005.
- [16] Hwang, J. Y., Lee, P. J. Efficient Broadcast Encryption with Log-Key Storage. *Cryptology ePrint Archive, Report 2005/373*, 2005.
- [17] Jho, N.-S., Hwang, J. Y., Cheon, J. H., Kim, M.-H., Lee, D. H., Yoo, E. S. One-way Chain Based Broadcast Encryption Scheme. *Advances in Cryptology: EUROCRYPT 2005 (LNCS 3494)*, pp. 559-574, 2005.
- [18] Ramkumar, M. On Broadcast Encryption Scheme with Random Key Pre-distribution Schemes. *ICISS 2005*, pp. 304-316, 2005.
- [19] Ramzan, Z., Woodruff, D. Fast Algorithms for the Free Riders Problem in Broadcast Encryption. *Advances in Cryptology: CRYPTO 2006*, 2006.