Fast Algorithms for the Free Riders Problem in Broadcast Encryption

Zulfikar Ramzan[1⋆] and David P. Woodruff[2,3⋆]

[1] Symantec, Inc. `zulfikar_ramzan@symantec.com`
[2] MIT `dpwood@mit.edu`
[3] Tsinghua University

Abstract. We provide algorithms to solve the *free riders* problem in broadcast encryption. In this problem, the broadcast server is allowed to choose some small subset F of the revoked set R of users to allow to decrypt the broadcast, despite having been revoked. This may allow the server to significantly reduce network traffic while only allowing a small set of non-privileged users to decrypt the broadcast.

Although there are worst-case instances of broadcast encryption schemes where the free riders problem is difficult to solve (or even approximate), we show that for many specific broadcast encryption schemes, there are efficient algorithms. In particular, for the complete subtree method [25] and some other schemes in the subset-cover framework, we show how to find the optimal assignment of free riders in $O(|R||F|)$ time, which is independent of the total number of users. We also define an approximate version of this problem, and study specific distributions of R for which this relaxation yields even faster algorithms.

Along the way we develop the first approximation algorithms for the following problem: given two integer sequences $a_1 \geq a_2 \geq \cdots \geq a_n$ and $b_1 \geq b_2 \geq \cdots \geq b_n$, output for all i, an integer j' for which $a_{j'} + b_{i-j'} \leq (1+\epsilon)\min_j(a_j + b_{i-j})$. We show that if the differences $a_i - a_{i+1}, b_i - b_{i+1}$ are bounded, then there is an $O(n^{4/3}/\epsilon^{2/3})$-time algorithm for this problem, improving upon the $O(n^2)$ time of the naive algorithm.

Keywords: free riders, broadcast encryption, applications

1 Introduction

Broadcast Encryption: Broadcast encryption schemes allow a single center to transmit encrypted data over a broadcast channel to a large number of users U such that only a select subset $\mathcal{P} \subseteq U$ of *privileged users* can decrypt it. Such schemes aim to provide one-to-many secure communication services to a large user base without incurring scalability costs. One important application of broadcast encryption is to provide the users in a group with a common cryptographic key that they can then use to communicate amongst each other.

⋆ Part of this research was done while both authors were at DoCoMo Labs.

Other traditional applications of broadcast encryption include secure multicast of privileged content such as premium video, audio, and text content as well as protection on external storage devices such as the hard disk of a mobile phone, USB storage devices, Flash cards, CD and DVD ROMs, etc.

Broadcast encryption schemes typically involve a series of pre-broadcast transmissions at the end of which the users in P can compute a broadcast session key bk. The remainder of the broadcast is then encrypted using bk. There are a number of variations on this general problem:

– Privileged Sets: Privileged sets may be determined arbitrarily, have fixed size, or contain some type of hierarchical structure. They can be static across all broadcasts or dynamic. In other words, a user may be revoked permanently or only have his privileges temporarily revoked for a particular broadcast.
– Key Management: User keys may be fixed at the onset, be updated each time period, or be a function of previous transmissions.
– Coalition Resistance: The scheme may be secure against any revoked user coalition, or there may be some bound on the size of the tolerable coalition.

The most widely applicable scenario (and the one we consider in this paper) is where the revocation set is dynamic, the users are only given a collection of set-up keys at the onset (these keys are used for the lifetime of the system), and the system can tolerate an arbitrary number of colluders.

The following performance parameters are of interest: the number of pre-broadcast transmissions t made by the center, the amount of keying material k the receivers must persistently store, and the amount of time τ_d the receiver must spend to derive the broadcast session key bk from the pre-broadcast transmissions. Let $R = U \setminus P$ be the set of revoked users, let $r = |R|$, and let $n = |U|$.

Related Work: Berkovits introduced the concept of broadcast encryption [6]. In his scheme $t = O(n)$, although $k = 1$; the scheme is also insecure if used repeatedly. Later, Fiat and Naor [14] formalized the basic problem. The topic has since been studied extensively (e.g., see [4, 7, 10, 15, 20, 21, 22, 23, 25, 17]). We limit our discussion to the work most relevant to us. Naor et al. [25] proposed the *Complete Subtree* (CS) scheme, which is stateless and requires a pre-broadcast transmission of $t = O(r \log \frac{n}{r})$ ciphertexts and storage of $k = O(\log n)$ keys per user. The scheme is simple and provides information-theoretic security. In [25], the *Subset Difference* (SD) scheme is also proposed. Here, t is reduced to $O(r)$, and is therefore independent of the total number of users n. On the other hand, the scheme only provides computational security and a user must store $k = O(\log^2 n)$ keys. Furthermore, the receiver must perform $\tau_d = O(\log n)$ computation to compute the broadcast key bk. Halevy and Shamir [17] improved upon this construction with their *Layered Subset Difference* (LSD) scheme. For any $\epsilon > 0$, they achieve $k = O(\log^{1+\epsilon} n)$ without substantially increasing t.

The CS, SD, LSD, are all examples of schemes in the subset-cover framework. Here various subsets of the users are formed and a cryptographic key is assigned to each subset. Each user is provided with all the keys for subsets he is a member of. To encrypt to a privileged set, the center finds a collection of subsets whose union is exactly the privileged set, and encrypts content using just those keys.

Free Riders in Broadcast Encryption: Most of the prior work on broadcast encryption assumes a stringent security requirement. In particular, any member of the privileged set $\mathcal{P}$ can decrypt a broadcast, but *no one* outside of this set can. For many applications this might be excessive. In particular, the system might be able to tolerate some number of non-privileged users being able to decrypt a broadcast. Such a user is termed a *free rider.*

Here the center chooses some (preferably small) subset F of R who can decrypt the broadcast, despite having been revoked. Let $f = |F|$. In general, this concept is useful in commercial applications of cryptography (e.g., by network operators) since the negative fiscal impact of allowing some small number of free riders may be dwarfed by the savings incurred by reducing other operational systems costs, such as the cost of transmitting traffic on the network, etc. We will demonstrate the benefits of choosing free riders in Section 3.2.

Abdalla, Shavit, and Wool (ASW) [1] introduced the notion of free riders. They demonstrate that allowing free riders can reduce the amount of data communicated in many situations. Next, they consider the problem of determining the free rider assignment that minimizes traffic for a given number of free riders. Via simple reductions from SetCover, they observe that this problem is NP complete in the worst case, and is unlikely to have a polynomial-time approximation scheme (PTAS). Further, they suggest some heuristic algorithms that apply to broadcast encryption schemes in the subset-cover framework. Finally, they experimentally analyze their heuristics on the CS scheme.

Our Contributions: In this paper we first show that for the CS scheme, one can provably obtain the *optimal solution* (with respect to reducing the overall traffic) to the free riders problem in worst-case time $O(rf + r\log\log n)$. Our techniques are also applicable to other schemes in the subset-cover framework.

In contrast, ASW [1] only give heuristics with no performance guarantees.[4] Thus, our work *provably* demonstrates the positive aspects of free riders in broadcast encryption. Moreover, our running time is almost *independent* of n, and as $r, f << n$ in practice, is likely to be extremely efficient. Note that neither us nor ASW consider the role of free riders on improving other aspects of performance, like the amount of storage required on end-user devices. In fact, our storage requirements are the same as that of the underlying broadcast encryption scheme, but the overall traffic is reduced.

Second, we relax the free riders problem to allow an algorithm to output a free rider assignment with cost at most $(1+\epsilon)$ times that of the optimal assignment.

[4] Even though our algorithms have provable performance guarantees, our work does not contradict the ASW NP-hardness or inapproximability results. In particular, their results only apply to worst-case instances. In our case, we give optimal solutions for *specific* set-covering schemes. Interestingly, our algorithms apply to the very same broadcast encryption scheme (the complete subtree method) that ASW used in experimentally analyzing their heuristics.

We show how this relaxation is likely to be useful for *specific distributions* of revoked users R. In particular, we show that when R is uniformly chosen from sets of size r, it results in a running time of $O(rf^{1/3}\text{polylog } n)$ for constant ϵ.

Third, consider the following MinSum problem: given two arrays of integers $a_1 \geq a_2 \geq \cdots \geq a_{m_1} \geq 0$ and $b_1 \geq b_2 \geq \cdots b_{m_2} \geq 0$ such that for all i, $a_i - a_{i+1}$ and $b_i - b_{i+1}$ are bounded by L, output an array c such that for all $2 \leq i \leq m_1 + m_2$, $a_{c[i]} + b_{i-c[i]} \leq (1+\epsilon)\min_{j=1}^{i-1}(a_j + b_{i-j})$. Note that $c[i]$ is an integer between 1 and $i-1$, inclusive. There is a trivial algorithm for this which runs in time $O(m_1 m_2)$. We show a direct connection between this problem and the running time of our algorithm for specific distributions on R. Using tools from computational geometry, the running time of MinSum was recently improved [8, 13] to $O(m_1 + m_2 + m_1 m_2 / \log(m_1 + m_2))$, which we can apply to our problem (for $\epsilon = 0$) to obtain faster solutions to the free riders problem.

More importantly, as far as we are aware, we provide the first approximation (e.g., $\epsilon > 0$) algorithms for MinSum which asymptotically beat the trivial $O(m_1 m_2)$ running time. Namely, we achieve $O(m_1^{1/3} m_2 L^{2/3}/\epsilon^{2/3})$ time, which for constant ϵ and small L (as is the case for the CS, SD, and LSD schemes), is about $m_1^{1/3} m_2$. This algorithm may be of independent interest since the MinSum problem occurs naturally in computational geometry, see, e.g., [13], where the problem is compared with computing all-pairs-shortest-paths.

Our algorithms extend to other broadcast encryption schemes that use set covering mechanisms with a hierarchical tree-like structure, such as the natural extension of the CS scheme to k-ary trees. We suspect they also extend to the SD [25] and LSD [17] schemes, and the natural extensions of these schemes to k-ary trees. Further, our techniques apply to multi-certificate revocation/validation and group key management, which we discuss below.

Other Applications of Free Riders: The idea of designing systems that permit some number of non-privileged users to receive the same services as privileged users to lower system costs is widely applicable. We mention other settings where the techniques in this paper apply (that do not seem to have been considered previously). One such application is multi-certificate revocation/validation [2]. Here a certificate authority (CA) in a public-key infrastructure might wish to revoke a number of user certificates, but could be willing to permit a subset of these users to get away with using revoked credentials. Multi-certificate revocation/validation schemes allow the CA to issue a single proof that validates the non-revoked certificates for a specified time period. Permitting free riders decreases the proof size, which reduces communication complexity.

Another application is dynamic group key management [3, 27, 19, 24, 30, 29], which is closely related to broadcast encryption. Here a user group communicates among each other using a common cryptographic key. As users join or leave the group at different times the key is updated by sending out a new (encrypted) group session key. If one were willing to allow a small number of former group members to be free riders and decrypt this key-update message, then the overall communication costs could be lowered. Popular schemes for multi-certificate revocation and group key management use the same set-covering mechanisms encountered in broadcast encryption, and thus our techniques apply.

Organization: Section 2 discusses preliminaries and gives an overview of our algorithm, while section 3 gives more detail. Section 4 discusses our algorithmic improvements to the MinSum problem. Due to space constraints, we defer many proofs to the full version of this paper [28].

2 Preliminaries and Overview of our algorithm

Preliminaries: Let U denote the users, with $n = |U|$. Let R denote the revoked set at a given time, with $r = |R|$. Let F denote the set of free riders with $f = |F|$.

We now define the complete subtree (CS) scheme [25], which falls into the subset-cover framework mentioned in Section 1. W.l.o.g., assume n is a power of 2. Users are associated with the n leaves of a complete binary tree. The server creates a key for each node v of the binary tree, and distributes it to all users in the subtree rooted at v. One can find $O(r \log n/r)$ keys (nodes) such that every user in $U \setminus R$ has one such key, but each user in R lacks all such keys [25].

Benefits of Free Riders: We give an example to demonstrate the benefits of choosing the optimal set of free riders for the CS scheme. Suppose $r = \sqrt{n}$ and $f = cr = c\sqrt{n}$ for some constant $0 < c < 1$. For simplicity suppose that $r, r - f$ are powers of 2. Consider the level L of the complete binary tree, as used in the CS scheme, with exactly $\sqrt{n}$ nodes. Put $r - f$ revoked users in a *complete* subtree of one node in level L, and for each other node L, put at most 1 revoked user (so exactly f nodes in L have at least one revoked user in their subtree). Then the optimal F contains the f isolated users. The revoked users not in F constitute all the leaves of a complete subtree, and we only need $O(\log n)$ traffic to transmit to the set $U \setminus (R \setminus F)$. This is accomplished by including a key at each sibling along the path from the root of the complete subtree to the root of the complete binary tree.

If, however, we choose the free riders randomly, then with overwhelming probability $\Omega(f)$ isolated revoked users (i.e., those not included in the complete subtree) remain. For each such user u, we must include a key at each sibling along the path from u to L (not including the node in level L). Since this path length is $\log n - \log \sqrt{n} = \Omega(\log n)$, and since the subtrees rooted in level L are disjoint, the total network traffic is $\Omega(f(\log n - \log \sqrt{n})) = \Omega(\sqrt{n} \log n)$. Therefore, by choosing F sensibly (as opposed to randomly) we save a $\sqrt{n}$ factor in network traffic.

The reader is referred to ASW [1] for a more extensive analysis of the benefits of free riders on the network traffic. Here we concentrate on the efficiency of algorithms for finding F.

Notation: For a vertex v in a tree, let $T(v)$ be the subtree rooted at v, and let $L(v)$ and $R(v)$ denote the subtrees rooted at the left and right child of v, respectively. We use a standard identification of nodes of a complete binary tree. Namely, we define the root to be 1. Then, for a node identified with integer i, its left child is $2i$ and its right child is $2i+1$. We define the height of a node v to be $\lfloor \log_2 v \rfloor$, so the root has height 0. For nodes u, v, we define $a(u,v)$ to be the common ancestor of u and v of largest height. We say that node u is *to the left* of node v if $u \in L(a(u,v))$ while $v \in R(a(u,v))$. Let log() denote the base-2 logarithm. We work in the RAM model where arithmetic operations on $O(\log n)$ size words take $O(1)$ time.

Cost Function: Let $F \subseteq R$ be subject to $|F| \leq f$. Let V be a subset of vertices of the complete binary tree for which

$$(R \setminus F) \cap \cup_{v \in V} T(v) = \emptyset \text{ and } (U \setminus R) \subseteq \cup_{v \in V} T(v).$$

Then the cost of F is the size of a minimal such V, and the optimal assignment F is that with the smallest cost. This cost function captures the network traffic in any scheme within the subset-cover framework, since in these schemes the number of ciphertexts the server transmits equals $|V|$.

We first show how to quickly compute an optimal assignment of free riders for the CS scheme. We focus on this scheme for its simplicity and to contrast our results with the original free riders paper [1]. For these schemes we prove the following about our main algorithm Freerider-Approx.

Theorem 1. *Algorithm* Freerider-Approx(R, f) *outputs an optimal assignment of f free riders in $O(rf + r \log\log n)$ time.*

Remark 1. Although we use the terminology Freerider-Approx, the algorithm we describe is also capable of producing an optimal answer (i.e., no approximation).

To appreciate the techniques of Freerider-Approx, we first sketch a simple dynamic-programming algorithm achieving $O(nf)$ time. This already shows that many natural broadcast encryption schemes admit theoretically-efficient (poly(n) time) algorithms for the free riders problem, despite the fact that the problem is NP-hard in general.

Recall that in the CS scheme, users are associated with leaves of a complete binary tree, and the revoked set R determines some set of r leaves. For each node v in the tree, we can use dynamic programming to compute the optimal traffic $opt(i, T(v))$ attained by assigning at most i free riders to its subtree $T(v)$, for each $0 \leq i \leq f$. Note that if $i \geq |T(v) \cap R|$ then either every leaf of $T(v)$ is in R, in which case $opt(i, T(v)) = 0$, else $opt(i, T(v)) = 1$ since we can just let everyone in $T(v)$ receive the broadcast using a single message. Now, suppose for each $0 \leq i \leq f$ we have $opt(i, L(v))$ and $opt(i, R(v))$, together with indicator bits stating whether $L(v) \cap R$ and $R(v) \cap R$ are empty. Then the optimal cost of assigning at most i free riders to $T(v)$ is 1 if both indicator bits are set. This is because, in this case, and only in this case, we can use a single key at v. In this case we set the indicator bit at v to be 1. Otherwise the cost is $\min_j (opt(j, L(v)) + opt(i-j, R(v)))$. We can find this collection of minima in $O(f^2)$ time, for $0 \leq i \leq f$. Finally, by tracking which indices realize the minima, we can backtrack to find the optimal free-rider assignment itself. This running time is $O(nf^2)$, which we can reduce to $O(nf)$ by observing that finding the minima for each v only requires $O(|L(v) \cap R| \cdot |R(v) \cap R|)$ time. Unfortunately, this running time depends linearly on n, which may be huge in practice.

On the other hand, r is likely to be small, which we can exploit by observing that it suffices to perform the dynamic programming step for only a very small set of internal nodes called the *meeting points*. These are defined as follows. Recall that a Steiner tree of a subset U of vertices of an unweighted tree is a minimal-sized tree containing each vertex of U. Suppose we compute the Steiner tree T of the subset R of the complete binary tree, where the ith revoked user is identified with the ith leaf. Then the *meeting points* of T are the leaves of T together with the internal nodes v of the binary tree for which both $L(v)$ and $R(v)$ are non-empty (i.e., they contain a revoked user). We refer to the latter nodes as *internal meeting points*. A simple induction shows there are $2r-1$ meeting points, $r-1$ of which are internal. We give a very efficient $O(r \log\log n)$-time algorithm to output the list of meeting points.

Now, if a node v is not a meeting point, then there is no reason to perform the dynamic programming step on v, as its optimal assignments can be immediately inferred from those of its only child. The next question is how to perform the dynamic programming step, which computes $c_i = \min_j(a_j + b_{i-j})$ for all i, where a_k, b_k, c_k are the smallest costs of assigning at most k free riders to the trees $L(v), R(v)$, and $T(v)$, respectively. This is an instance of the MinSum problem. Suppose v is the meeting point joining meeting points x and y, and let $r_x = |L(v) \cap R|$ and $r_y = |R(v) \cap R|$. Then there is an $O(\min(f, r_x)\min(f, r_y))$ time algorithm for the MinSum problem, which works by computing sums $a_j + b_{i-j}$ for all i, j, and then taking the appropriate minima.

In fact, MinSum can be solved in time

$$O\left(\min(r_x, f) + \min(r_y, f) + \frac{\min(r_x, f)\min(r_y, f)}{\log(\min(f, r_x) + \min(f, r_y))}\right)$$

using ideas from computational geometry [8, 13]. It turns out that there are two properties of our problem which allow us to significantly reduce this running time. The first is that for the schemes we consider, $a_i - a_{i+1}$ and $b_i - b_{i+1}$ are very small (i.e., $\log n$ for for the CS scheme and $O(1)$ for the SD scheme). The second is that for many applications, it suffices to output an *approximate* solution to the free riders problem, that is, an assignment of at most f free riders with resulting cost at most $(1+\epsilon)$ times that of the optimal. To this end, we relax MinSum to just require that we output an array d with $d_i \leq (1+\epsilon)c_i$ for all i.

We show in section 3.1 that if $a_i - a_{i+1}, b_i - b_{i+1}$ are bounded by L, we can output d in $O((r_x)^{1/3} r_y L^{2/3}/\epsilon^{2/3})$ time. For the small values of L that we consider, this shows that a constant-factor approximation can be computed in about $(r_x)^{1/3} r_y$ time, significantly improving the time of the naive algorithm.

Unfortunately, it turns out that for a worst-case choice of R, our improvements for MinSum do not asymptotically improve the running time of Freerider-Approx, though they do so for *specific distributions* of R that may occur in practice. In particular, for $f = \Omega(\log n)$ we show that when R is uniformly chosen from sets of size r, then with probability at least $1 - 1/n$ Freerider-Approx outputs a constant-factor approximation in $O(rf^{1/3}\text{polylog } n)$ time, whereas if Freerider-Approx were to use the naive MinSum algorithm, its running time would be $\Omega(rf)$. The more general version of our theorem, for arbitrary f and arbitrary approximation ratios, can be found in Section 3.

For a worst-case analysis, we analyze Freerider-Approx in terms of the naive algorithm for MinSum and show by a careful choice of parameters that the algorithm outputs a $(1 + \epsilon)$-approximation. Showing that it only takes $O(rf)$ time requires an analysis over Steiner trees, which although elementary, is non-trivial.

3 Details of the algorithm

3.1 Finding the meeting points

We assume $0 < f < r < n$, as otherwise the problem is trivial. Thus, we can root T at an internal meeting point. We root T at the meeting point of smallest height in the complete binary tree. This node is unique, since if there were two such nodes their common ancestor would be a meeting point of smaller height.

We assume that we are given R as a list of integers in the range $\{n, n + 1, \ldots, 2n - 1\}$. We first sort R. The best known integer sorting algorithms run in expected time $O(r\sqrt{\log \log r})$ [18] and worst-case $O(r \log \log r)$ [16], though more practical ones running in time $O(r \log r)$ or $O(r \log \log n)$ can be based on textbook algorithms [9] or van Emde Boas trees [11, 12].

One way to visualize our algorithm for computing the meeting points is to think of a line containing r points. The ith point is u_i, the ith revoked user in the given sorted list. The edge (u_i, u_{i+1}) is labeled as follows: we find $a = a(u_i, u_{i+1})$, and we label (u_i, u_{i+1}) with both a and $\mathsf{height}(a)$, referring to the latter as the *edge weight*. Note that a and its height can be found using $O(\log \log n)$ arithmetic operations via a binary search on the binary representations of u_i and u_{i+1} (recall that we are assuming the unit-cost RAM model). Then a is an internal meeting point with $u_i \in L(a)$ and $u_{i+1} \in R(a)$.

We maintain a sorted list E of edge weights and an output list MP of meeting points. We initialize MP to R. We then find the largest-weight edge $e = \{u, v\}$ (i.e., $a(u, v)$ is of largest height), breaking ties arbitrarily, add $a(u, v)$ to the end of MP, and delete e from E. This corresponds to collapsing the edge $\{u, v\}$ on the line and replacing it with the vertex $a(u, v)$. We then update the weight and ancestor label of the other edges in E containing u or v (e.g., the neighbors of the collapsed edge on the line). We show that the "line structure" of the remaining edges E is preserved across iterations, so any vertex u on the line can belong to at most 2 edges, and thus this update step is efficient. After $r - 1$ deletions, the entire line is collapsed into a single vertex, the root of the Steiner tree, and we are done. The running time is $O(r \log \log n)$ plus the time to sort E.

A list of meeting points is *ordered* if for each meeting point v, its unique closest meeting points (under the shortest path distance) p_l and p_r, in $L(v)$ and $R(v)$ respectively, precede v in the list. In the full version of this paper [28], we prove:

Theorem 2. Meeting-Points *outputs an ordered list of meeting points in time* $O(r \log \log n)$.

3.2 The free rider approximation algorithm

Our free rider approximation algorithm Freerider-Approx makes use of the subroutine MeetingPoints and also the subroutine Min-Sum. We defer the description of Min-Sum to Section 4, but we summarize its relevant properties as follows.

Theorem 3. *Let* $\epsilon \geq 0$, *and let* $a_1 \geq a_2 \geq \cdots \geq a_{m_1} \geq 0$ *and* $b_1 \geq b_2 \geq \cdots \geq b_{m_2} \geq 0$ *be integer sequences such that for all* i, $a_i - a_{i+1}$ *and* $b_i - b_{i+1}$ *are bounded by* L. *Then* $\text{Min-Sum}((a_i),(b_i),\epsilon)$ *outputs an array* c *such that for all* $2 \leq i \leq m_1 + m_2$, *we have* $a_{c[i]} + b_{i-c[i]} \leq (1+\epsilon)\min_j\ (a_j + b_{i-j})$.

1. *If* $\epsilon > 0$, *the time complexity is* $O\left(\frac{m_1^{1/3} m_2 L^{2/3}}{\epsilon^{2/3}}\right)$.
2. *If* $\epsilon = 0$, *the time complexity is* $O\left(\frac{m_1 m_2}{\log(m_1+m_2)} + m_1 + m_2\right)$.

The basic idea behind our algorithm Freerider-Approx for computing an assignment of free riders is to use dynamic programming on the ordered list of meeting points provided by MeetingPoints, invoking Min-Sum for each dynamic programming step. Note that MeetingPoints returns a list of *ordered* meeting points (see Section 3.1), so we can do dynamic programming by walking through the list in order. This effectively replaces the factor of n in the time complexity of the naive dynamic programming algorithm with a factor of r.

Now consider the approximation error. Suppose we want a free-rider assignment whose resulting traffic cost is at most $(1+\epsilon)$ times that of the optimal assignment. Suppose for each dynamic programming step we invoke Min-Sum with parameter $\epsilon' = \epsilon/(2\min(r, \log n))$. When we combine two such approximations, we obtain a $(1+\epsilon')^2$ approximation. Since any path in the Steiner tree contains at most $\min(r, \log n)$ meeting points, the total error is at most $(1+\epsilon)$.

For each internal meeting point v, we assume the two meeting points x, y for which $v = a(x,y)$ can be determined in constant time. This is easily achieved by keeping track of pointers in the implementation of MeetingPoints. Let $\text{MPleft}(v)$ be the operation returning x, and $\text{MPright}(v)$ the operation returning y.

In our algorithm each meeting point u has data fields r_u and A_u. Here r_u denotes the number of revoked users in the tree $T(u)$. A_u is an array, indexed from $0 \leq i \leq \min(f, r_u)$, which records the cost found by our algorithm of assigning at most i free riders to $T(u) \cap R$. Finally, for internal meeting points u, there is a field B_u which is a companion array to A_u, indexed from $0 \leq i \leq \min(f, r_u)$, such that $A_u[i] = A_{\mathsf{MPleft(u)}}[B_u[i]] + A_{\mathsf{MPright(u)}}[i - B_u[i]]$. Thus, B_u records the assignments of free riders realizing the costs found by A_u.

By a simple inductive argument (given in the full version), one can show that $|MP| = 2r - 1$. Our main algorithm is described below.

Freerider-Approx (R, f, ϵ):

1. $MP \leftarrow \mathsf{MeetingPoints}(R)$, $\epsilon' = \epsilon/(2\min(r, \log n))$.
2. For each revoked user u, set $r_u = 1$ and $A_u \leftarrow (0, 0)$.
3. For $i = r + 1$ to $2r - 1$,
   (a) $v \leftarrow MP[i]$, $x \leftarrow \mathsf{MPleft}(v)$, $y \leftarrow \mathsf{MPright}(v)$.
   (b) Add $\mathsf{height}(x) - \mathsf{height}(v) - 1$ to each entry of A_x.
   If $r_x \leq f$ and $A_x[r_x] \neq 0$, set $A_x[r_x] = 1$.
   (c) Add $\mathsf{height}(y) - \mathsf{height}(v) - 1$ to each entry of A_y.
   If $r_y \leq f$ and $A_y[r_y] \neq 0$, set $A_y[r_y] = 1$.
   (d) $B_v \leftarrow \mathsf{MinSum}(A_x, A_y, \epsilon')$.
   (e) $r_v = r_x + r_y$.
   (f) For each $0 \leq i \leq \min(f, r_v)$, set $A_v[i] = A_x[B_v[i]] + A_y[i - B_v[i]]$.
   If $r_v \leq f$ and $A_v[r_v] \neq 0$, set $A_v[r_v] = 1$.
4. Output $A_{MP[2r-1]}[f] + \mathsf{height}(MP[2r - 1])$ as the cost.
5. Use the B_v's to do a Depth-First-Search on MP to find/output the assignment.

Due to space constraints, we defer the formal proof of the following theorem to the full version of the paper [28], though we give some intuition and bound its time complexity here.

Theorem 4. Freerider-Approx *outputs an assignment of free riders with cost at most* $(1 + \epsilon)$ *times that of the optimum.*

The algorithm can be explained as follows. For each revoked user u, if we assign no free riders to $T(u) = u$, then $T(u)$'s cost is 0 since $T(u)$ contains no privileged users. Thus the cost of assigning at most one free rider to $T(u)$ is also 0, so in step (2) we set $A_u \leftarrow (0, 0)$, consistent with the definition of A_u.

In step (3b) or (3c), it may be the case that x or y is not a child of v. Suppose this is true of x. Now consider the optimal assignment of at most i free riders to $L(v)$ for some i. Then, unless $r_x \leq f$, there must be a revoked user in $T(x)$ that cannot be made a free rider. Let P be the shortest-path from x to v, not including v. It is then easy to see that the optimal assignment of at most i free riders to $L(v)$ is the optimal assignment of at most i free riders to $T(x)$ together with one extra key for each sibling of a node on P. Thus, we should add the length of P to each entry of A_x, and $|P| = \mathsf{height}(x) - \mathsf{height}(v) - 1$. This logic also explains step (4). Indeed, in this case we consider the path from $\mathsf{MP}[2r - 1]$ to the root of the complete binary tree, which has length $\mathsf{height}(MP[2r - 1])$.

There are two exceptions though. The first is if $r_x \leq f$, in which case we should reset $A_x[r_x]$ to 1 since we can turn the r_x revoked users into free riders and use a single key at the root of $L(v)$ to cover all of v. There is, however, one more case for which this may not be optimal. This occurs if $A_x[r_x] = 0$ (after adding $\mathsf{height}(x) - \mathsf{height}(v) - 1$ to each entry of A_x) in which case every leaf of $T(x)$ is a revoked user, and thus there is 0 cost if we do not assign any free riders. This can only occur if x is in fact a child of v. In this case, we should leave $A_x[r_x] = 0$. This logic also explains the if statement in step (3f).

We now elaborate on step (5). If, for instance, v is the root of the Steiner tree with $x = \mathsf{MPleft(v)}$ and $y = \mathsf{MPright(v)}$ with $B_v[f] = i$, then one assigns at most i free riders to $T(x)$ and at most $f - i$ to $T(y)$. If $A_x[i] = 0$, it means that the leaves of $T(x)$ are all revoked users, and we should not assign any of them to be free riders. Otherwise, if $i \geq r_x$, we make every revoked user in $T(x)$ a free rider. Otherwise, we recurse on MPleft(x) and MPright(x). The analysis for y is similar, and it is easy to see the overall time complexity of this step is $O(r)$.

Let us now look at the time complexity of Freerider-Approx. We first note that as we have stated the algorithm, we cannot hope to do better than $O(rf)$.

Lemma 1. *For any ϵ, the worst-case time complexity of* Freerider-Approx *in the unit-cost RAM model is $\Omega(rf)$.*

Proof. Suppose the revoked set has the form

$$u_1 = 2n - 1, u_2 = 2n - 2, u_3 = 2n - 4, u_4 = 2n - 8, \ldots, u_i = 2n - 2^{i-1}, \ldots$$

In this case the Steiner tree consists of vertices $u_1, \ldots, u_r, v_1, \ldots, v_{r-1}$, where the v_i are the internal meeting points satisfying, $L(v_i) \cap R = u_{i+1}$ and $R(v_i) \cap R = \{u_1, u_2, \ldots, u_i\}$. Note that the time complexity of Min-Sum is at least $|A_y|$ for any ϵ. Then in this example, for $i \geq f$, there are at least f revoked users in $R(v_i)$, and thus the time complexity of MinSum at meeting point v_i is at least f. It follows that the time complexity of Freerider-Approx is at least $\Omega((r - f)f) = \Omega(rf)$.

It turns out that the above lemma is indeed a worst-case.

Theorem 5. *The time complexity of* Freerider-Approx *is $O(rf + r \log \log n)$.*

Proof. By Theorem 2, steps (1-2) can be done in $O(r \log \log n)$ time. The time complexity of $\mathsf{MinSum}(A_x, A_y, \epsilon')$ is at least $|A_x| + |A_y|$, so the time complexity of step (3) is dominated by step (3d). Note also that step (4) can be done in $O(1)$ time and step (5) in $O(r)$ time, so the total time of the algorithm is $O(r \log \log n)$ plus the total time spent in step (3d). If $C(|A_x|, |A_y|)$ denotes the time complexity of $\mathsf{MinSum}(A_x, A_y, \epsilon')$, then the total time spent in step (3d) is

$$\sum_{i=r+1}^{2r-1} C(|A_{\mathsf{MPleft}\ (MP[i])}|, |A_{\mathsf{MPright}\ (MP[i])}|). \tag{1}$$

To bound this sum, observe that for any v, $|A_v| = 1 + \min(f, r_v)$. To solve MinSum, we will consider the naive $O(m_1 m_2)$-time algorithm, as it turns out in the *worst-case* our faster algorithms of Section 4 do not yield an asymptotic improvement. Then, up to a constant factor, sum (1) is

$$\sum_v \min(f, r_{\mathsf{MPleft}(v)}) \min(f, r_{\mathsf{MPright}(v)}), \tag{2}$$

where the sum is over all internal meeting points v.

It will be convenient to define the *collapsed Steiner tree* on the revoked set R. This tree is formed by taking the rooted Steiner tree on R, and then repeatedly collapsing edges for which either endpoint is not a meeting point (if one endpoint is a meeting point, the collapsed edge is labeled by that endpoint). We are left with a binary tree on $2r-1$ nodes (the meeting points) containing r leaves (the revoked users). Denote this tree by T.

We first consider the contribution S to (2) from v with $r_{\mathsf{MPleft(v)}}, r_{\mathsf{MPright(v)}} < f$, i.e.,

$$S = \sum_{v \ | \ r_{\mathsf{MPleft(v)}}, r_{\mathsf{MPright(v)}} \ < \ f} \min(f, r_{\mathsf{MPleft}(v)}) \min(f, r_{\mathsf{MPright}(v)})$$

$$= \sum_{v \ | \ r_{\mathsf{MPleft(v)}}, r_{\mathsf{MPright(v)}} \ < \ f} r_{\mathsf{MPleft}(v)} r_{\mathsf{MPright}(v)}.$$

Create a formal variable X_u for each $u \in R$. Then by writing $|L(v) \cap R|$ as $\sum_{u \in L(v) \cap R} X_u$ and $|R(v) \cap R|$ as $\sum_{u \in R(v) \cap R} X_u$, we may express S as a degree-2 multilinear polynomial $S(\{X_u\}_{u \in R})$, where the actual value S corresponds to setting $X_u = 1$ for all u. We claim that in $S(\{X_u\}_{u \in R})$, for each u there can be at most $f-1$ different v for which $X_u \cdot X_v$ appears, and further, the coefficient of each of these monomials is 1.

To see this, let $a \in T$ be the ancestor of u of maximal height for which $r_{\mathsf{MPleft(a)}}, r_{\mathsf{MPright(a)}} < f$. Consider the path $u = u_0, u_1, \ldots, u_k = a$ from u to a in T. Let $v_0, v_1, \ldots, v_{k-1}$ be the siblings of $u_0, u_1, \ldots, u_{k-1}$. Then the subtrees $T(v_i)$ and $T(v_j)$ are disjoint for all $i \neq j$. Thus, $X_u \cdot X_v$ can appear at most once in S for a given $v \in T(a)$, and the number of such v is bounded by $r_a < 2f$.

As there are r different u, setting each X_u to 1 shows that $S = O(rf)$.

Next, consider the subgraph T' of T on nodes v for which either $r_{\mathsf{MPleft(v)}} \geq f$ or $r_{\mathsf{MPright(v)}} \geq f$ (or both). Observe that T' is a tree.

Consider any vertex v for which exactly one of $r_{\mathsf{MPleft(v)}}, r_{\mathsf{MPright(v)}}$ is at least f. In this case we say that v is *lop-sided*. Suppose $c(v)$ is v's child for which $r_{c(v)} < f$. Then the contribution from v to (2) is $f \cdot |T(c(v)) \cap R|$. We claim that $T(c(v)) \cap T(c(v')) = \emptyset$ for all lop-sided $v \neq v'$. Indeed, as T' is a tree, there is a path P from v to v' in T'. Moreover, as both $T(c(v))$ and $T(c(v'))$ are trees, if $T(c(v)) \cap T(c(v')) \neq \emptyset$, then there is a path P' from $c(v')$ to $c(v)$ in $T(c(v)) \cup T(c(v'))$. As T' and $T(c(v)) \cup T(c(v'))$ are disjoint, the path $Pc(v')P'v$ is a cycle in T, contradicting that T is a tree. Therefore,

$$\sum_{lop-sided \ v} \min(f, r_{\mathsf{MPleft(v)}}) \min(f, r_{\mathsf{MPright(v)}})$$

$$= \ f \sum_{lop-sided \ v} |T(c(v)) \cap R| \ \leq \ f \cdot r.$$

Thus, to bound (2), it remains to account for those vertices v for which both $r_{\mathsf{MPleft(v)}} \geq f$ and $r_{\mathsf{MPright(v)}} \geq f$. Observe that each such v contributes $O(f^2)$ to (2). So, if we can show that there are at most $O(r/f)$ such v, it will follow that (2) is bounded by $O(rf)$.

To this end, let T'' be the subgraph of T consisting of all vertices v for which $r_{\mathsf{MPleft}(v)} \geq f$ and $r_{\mathsf{MPright}(v)} \geq f$. Next, create a binary tree T''' by first adding T'' to T''', then adding $\mathsf{MPleft}(v)$ and $\mathsf{MPright}(v)$ to T''' for each $v \in T''$. Now, for any two leaves $v_1 \neq v_2$ in T''', we have $T(v_1) \cap T(v_2) = \emptyset$, as otherwise T would contain a cycle. Moreover, by definition of the vertices in T'', we have $|T(v_1)| \geq |T(v_1) \cap R| \geq f$ and $|T(v_2)| \geq |T(v_2) \cap R| \geq f$. Thus, there can be at most r/f leaves in T'''. As T''' is a binary tree, it follows that it can contain at most $2r/f = O(r/f)$ nodes, and thus T'' also contains at most this many nodes. This completes the proof.

Thus, we have proven Theorem 1. We now consider the case when the revoked set R is chosen uniformly at random from sets of size r. This might often be the case in practice, if say, the revoked users are uncorrelated with each other. This example exploits our algorithmic improvements in Section 4. In the full version of the paper [28], we prove the following theorem.

Theorem 6. *If the revoked set R is uniformly chosen from sets of size r, then with probability at least $1 - 1/n$, if* Freerider-Approx *uses our* MinSum $(1+\epsilon)$*-approximation algorithm, then it runs in time*

$$O\left(\frac{1}{\epsilon^{2/3}}\left(r\log^{8/3} n \;+\; rf^{1/3}\log^{5/3} n\right)\right),$$

which is $O(rf^{1/3}\text{polylog } n)$ for $\epsilon > \frac{1}{\text{polylog} n}$. On the other hand, if $r \geq \frac{4}{3}\log n$, then with probability at least $1-1/n$, if Freerider-Approx *uses the naive algorithm for* MinSum*, then it takes time*

$$\Omega\left(r\min\left(f, \frac{f^2}{\log n}\right)\right),$$

which is $\Omega(rf)$ for $f = \Omega(\log n)$.

The intuition behind the proof is as follows. For a randomly chosen R of size r, one can use a Chernoff-type bound for negatively-correlated random variables [26] to show that with probability at least $1-1/n$, for all nodes v in the complete binary tree, $|T(v) \cap R| \leq 2t(v)\frac{r}{n} + 2\log n$, where $t(v)$ is the number of leaves in $T(v)$. Note that, $\mathbf{E}[|T(v) \cap R|] = t(v)\frac{r}{n}$, so this variable is tightly concentrated.

To get the lower bound of $\Omega(rf)$ for the naive MinSum algorithm, we first consider nodes with height $\lfloor \frac{r}{2f} \rfloor$, assuming first that $f \leq r/4$. Using the above bound on $|T(v) \cap R|$, one can show that $\Omega(\frac{r}{f})$ of them must have $|T(v) \cap R| \geq f$, as otherwise there is some v with height $\lfloor \frac{r}{2f} \rfloor$ for which $|T(v) \cap R|$ is too large. It follows that there are $\Omega(\frac{r}{f})$ nodes with height less than $\lfloor \frac{r}{2f} \rfloor$ each contributing $\Omega(f^2)$ to sum (2), giving total cost $\Omega(rf)$. If $f > \frac{r}{4}$, one can show the root of the Steiner tree itself contributes $\Omega(rf)$ to sum (2).

Finally, to get the upper bound of $O(rf^{1/3}\text{polylog } n)$, one modifies sum (2) to,

$$\sum_v \min(f, r_{\mathsf{MPleft(v)}})^{1/3} \min(f, r_{\mathsf{MPright(v)}}) \left(\frac{L\log n}{\epsilon}\right)^{2/3},$$

where $L = \max_{i,v} opt(i, T(v)) - opt(i+1, T(v))$. One can use the bound on $|T(v) \cap R|$ above for each v to bound this sum. Using the definition of the CS scheme, it is easy to show that $opt(i, T(v)) - opt(i+1, T(v)) \leq \log n$ for any i, v.

Finally, we note that it is straightforward to extend Freerider-Approx to broadcast encryption schemes based on complete k-ary trees for general k. The idea is to binarize the tree by replacing a degree-k node with a binary tree on k nodes, and then run Freerider-Approx on the virtual tree.

We suspect that one can extend Freerider-Approx to the SD scheme of [25] with the same time complexity, though we have not worked out the details. The only change is the dynamic programming step, steps (3-4), for which one must be careful if a key involving the joining node is part of an optimal assignment of at most i free riders at a given subtree. One can run MinSum as before, but needs to compare the output to a few extra cases involving the key at the joining node (which MinSum does not account for).

4 Min-Sum problem

In this section we give our MinSum algorithms and analyses. Recall that we are given $\epsilon \geq 0$, and integer sequences $a_1 \geq a_2 \geq \cdots \geq a_{m_1} \geq 0$ and $b_1 \geq b_2 \geq \cdots \geq b_{m_2} \geq 0$ with $a_i - a_{i+1}, b_i - b_{i+1} \leq L$ for all i. The goal is to output an array c such that for all $2 \leq i \leq m_1 + m_2$, we have $a_{c[i]} + b_{i-c[i]} \leq (1+\epsilon) \min_j (a_j + b_{i-j})$. W.l.o.g., we will assume that $m_1 \leq m_2$. Thus, $m_1 + m_2 = O(m_2)$. The naive algorithm for this problem which, for all i, computes all pairs of sums $a_j + b_{i-j}$, runs in $O(m_1(m_1+m_2)) = O(m_1 m_2)$ time. We now show how to do much better.

4.1 Algorithm overview

The essence of our algorithm is the following balancing technique. The first idea is that, since the lists a_j and b_j are sorted, we can quickly find (via binary search) the largest "level" i^* for which $\min_j(a_j + b_{i^*-j}) \geq s$ for some parameter s to be optimized. Our algorithm then has two parts: its solution to levels $i \leq i^*$ and its solution to levels $i > i^*$.

To solve levels $i \leq i^*$, we have two ideas. For these levels, we crucially exploit the fact that the differences $a_j - a_{j+1}$ and $b_j - b_{j+1}$ are bounded by L and that $\epsilon > 0$. The first idea is that to solve some such level i, instead of computing all sums $a_j + b_{i-j}$ we can get by with only computing a small fraction of sums. Indeed, since the differences are bounded, we have $a_{j-1} + b_{i-(j-1)}$ is approximately equal to $a_j + b_{i-j}$ for all j. Since we only need an approximate answer, we can just take the minimum found over the small fraction of sums that we compute. This fraction depends on ϵ, L, and s. Note, the dependence on s arises since we want a *multiplicative approximation*, and thus, the smaller s is, the larger the fraction of sums we will need to compute.

The next idea is that we do not even need to solve all the levels $i \leq i^*$. Indeed, since the differences are bounded and we are content with an approximation, if we solve some level i' then we can use this solution for levels "close" to i', i.e., those levels in a range $[i'-d, i'+d]$ for some parameter d that depends on ϵ, L, and s. In other words, if $a_j + b_{i'-j}$ is our solution for level i', then for a level $i \in [i'-d, i'+d]$ we can bound its cost by $a_j + b_{i'-j}$.

To solve levels $i > i^*$ we have one main idea, which allows us to compute the *exact* solutions for these levels. Suppose $(j^*, i^* - j^*)$ realizes the optimum in level i^*. Then $a_{j^*} + b_{i^*-j^*} \geq s$. It follows that a_{j^*} and $b_{i^*-j^*}$ must both be less than $s + L$. Indeed, since the a_j and b_j are non-negative and the differences are bounded by L, then if say, $a_{j^*} \geq s + L$, then $a_{j^*+1} \geq s$ and thus the minimum in level $i^* + 1$ is at least s, contradicting the maximality of i^*. Thus, in the sequences $a_{j^*} \geq a_{j^*+1} \geq \cdots \geq a_{m_1}$ and $b_{i^*-j^*} \geq b_{i^*-j^*+1} \geq \cdots \geq b_{m_2}$ there are a lot of identical elements, i.e., we cannot have $a_\ell > a_{\ell+1}$ too often since on the one hand $a_{j^*} \leq s + L$, and on the other hand $a_{m_1} \geq 0$ (similarly for the b_j).

Then the key observation is that for any level $i > i^*$, the minimum value $a_j + b_{i-j}$ satisfies the following property:

$$\text{Either } j = j^*, \text{ or } a_{j-1} > a_j, \text{ or } b_{i-j-1} > b_{i-j}.$$

Indeed, suppose not, i.e., $a_{j-1} = a_j$ and $b_{i-j-1} = b_{i-j}$ for some $j \neq j^*$. Then, since $i > i^*$, either $j > j^*$ or $i - j > i - j^*$. W.l.o.g., suppose that $j > j^*$. Then, since the b_j values are non-increasing, we have that $a_{j-1} + b_{i-j+1}$ is also a minimum for level j. Repeating this "pushing" argument, we can push a_j until j either equals j^* or satisfies $a_{j-1} > a_j$. This is a contradiction.

Now as observed before, there cannot be too many different $j > j^*$ for which $a_{j-1} > a_j$. In fact, there can be at most $s + L$ of them. So we can just compute all valid pairs of sums in this case, and take the appropriate minima.

Finally, we choose s so as to balance the time complexities of these two parts.

4.2 Algorithm details

First consider $\epsilon > 0$. Our main algorithm MinSum will call upon a subroutine $\mathsf{MinSumLevel}((a),(b),i)$ which outputs $\min_j(a_j + b_{i-j})$ and an index j realizing this minimum. $\mathsf{MinSumLevel}$ can be implemented in $O(m_1)$ time by computing all the sums in the ith level and taking a minimum.

We need the notion of a *hop*. We say that (a_i, a_{i+1}) is a hop if $a_i > a_{i+1}$. In this case we say that a_{i+1} is a *hop-stop*. We similarly define hops and hop-stops for the sequence (b). For any integer z, if $a_i \leq z$, then since $a_{m_1} \geq 0$, there can be at most z hops in the sequence $a_i \geq a_{i+1} \geq a_{i+2} \geq \cdots \geq a_{m_1}$.

Let s be an integer parameter to be determined. MinSum first performs a binary search on level i to find the largest i for which $\min_j(a_j + b_{i-j}) \geq s$ (which works since the sequence $M_i = \min_j(a_j + b_{i-j})$ is non-increasing). This can be done in $O(m_1 \log(m_1 + m_2)) = O(m_1 \log m_2)$ time by using $\mathsf{MinSumLevel}$ at each level of a binary search on i. Let i^* be the largest such level, and let j^* be the index $\mathsf{MinSumLevel}((a),(b),i^*)$ returns realizing this minimum. By the maximality of i^*, it follows that $a_{j^*}, b_{i^*-j^*} \leq s + L$, so there can be at most $s + L$ hops to the right of a_{j^*} (including a_{j^*}), and at most $s + L$ hops to the right of $b_{i^*-j^*}$ (including $b_{i^*-j^*}$). Suppose $p_1 < p_2 < \cdots < p_\alpha$ are the indices of the hop-stops to the right of a_{j^*}, and let $q_1 < q_2 < \cdots < q_\beta$ be the indices of the hop-stops to the right of $b_{i^*-j^*}$. Note that $\alpha, \beta \le s + L$.

For levels $i > i^*$, MinSum computes all sums $a_x + b_y$ with $x \in \{j^*\} \cup \{p_1, \ldots, p_\alpha\}$ and $y \in [m_2]$, together with all sums $a_x + b_y$ with $x \in [m_1]$ and $y \in \{i^* - j^*\} \cup \{q_1, \ldots, q_\beta\}$. The number of sums computed is $O((s + L)(m_1 + m_2)) = O((s + L)m_2)$. Each sum is labeled by the level it falls in, and for each level, the minimum value is computed. Note that this can be done in $O((s+L)m_2)$ time simply by walking through the pairs and keeping track of the minimum for each level. Thus, in $O((s + L)m_2)$ time we have solved all levels greater than i^*.

Now let ϵ' be $\Theta(\epsilon s/(Lm_1))$, and fix any level $i \le i^*$. Then to solve level i exactly, there are at most m_1 pairs of indices we must take a minimum over:

$$(z, i - z), (z + 1, i - z - 1), (z + 2, i - z - 2), \ldots,$$

where $z = 1$ if $i \le m_2 + 1$, and otherwise $z = i - m_2$.

Instead of computing all these sums, we only compute those within the list:

$$(z,\ i - z), (z + \lfloor \epsilon' m_1 \rfloor,\ i - z - \lfloor \epsilon' m_1 \rfloor), (z + \lfloor 2\epsilon' m_1 \rfloor,\ i - z - \lfloor 2\epsilon' m_1 \rfloor),$$
$$(z + \lfloor 3\epsilon' m_1 \rfloor,\ i - z - \lfloor 3\epsilon' m_1 \rfloor),\ \ldots$$

We then take a minimum only over the sums with indices in this list. Note that for all i, $\lfloor (i + 1)\epsilon' m_1 \rfloor - \lfloor i\epsilon' m_1 \rfloor \le \epsilon' m_1 + 1$. Since $a_j - a_{j+1}, b_j - b_{j+1} \le L$, it follows that one of the sums from this list will be at most $\epsilon' m_1 L \le \epsilon s/2$ (for an appropriate choice of ϵ') more than the value of the true minimum for level i. Moreover, the number of sums computed is only $O(1/\epsilon')$, so the total time for level i is $O(Lm_1/(\epsilon s))$.

Finally, we observe that we don't have to solve every level $i \le i^*$, but rather we can use the output for level i as the output for levels "close" to i. Let ϵ'' be $\Theta(\epsilon s/(Lm_2))$. We want to solve levels $1, 2, \ldots, i^*$. Suppose we obtain an additive $\epsilon s/2$ approximation for each level in the following list:

$$1,\ \lfloor \epsilon'' i^* \rfloor,\ \lfloor 2\epsilon'' i^* \rfloor,\ \lfloor 3\epsilon'' i^* \rfloor,\ \ldots,\ i^*.$$

To output the minimum for some level i not appearing in the list, we find the nearest level $i' \ge i$ in the list. Then, suppose $a_j + b_{i'-j}$ is our solution for level i'. Then in constant time, we can find integers $d_1 \le j$ and $d_2 \le i' - j$ such that $d_1 + d_2 = i$. We can then output the pair (d_1, d_2) as our solution for level i (or more precisely, we just output d_1 since d_2 can be inferred). Since the sequences (a) and (b) are non-increasing, $a_{d_1} + b_{d_2} \le a_j + b_{i'-j}$.

Note that i' is at most $O(\epsilon'' i^*)$ levels away from i, so its minimum can be at most an additive

$$O(\epsilon'' L i^*) = O(\epsilon'' L(m_1 + m_2)) \le \epsilon s/2$$

more than that of i (for an appropriate choice of ϵ''). Thus, as the error in level i' is at most $\epsilon s/2$, and the minimum in level i' is at most $\epsilon s/2$ larger than that of level i, our output has additive error at most ϵs. Since the minimum in all levels $i \le i^*$ is at least s, our output is a $(1 + \epsilon)$-approximation.

Note that we need only solve $O(1/\epsilon'') = O(Lm_2/(\epsilon s))$ levels, each taking $O(Lm_1/(\epsilon s))$ time, so the time over all levels $i \leq i^*$ is $O(L^2 m_1 m_2/(\epsilon^2 s^2))$. Thus, the total time is $O(L^2 m_1 m_2/(\epsilon^2 s^2) + (s+L)m_2 + m_1 \log m_2)$. Setting $s = \Theta(m_1^{1/3} L^{2/3}/\epsilon^{2/3})$ gives total time complexity

$$O\left(\frac{m_1^{1/3} m_2 L^{2/3}}{\epsilon^{2/3}} + Lm_2\right).$$

The first term dominates this expression whenever $L \leq m_1$. On the other hand, if $L > m_1$, then $m_1^{1/3} m_2 L^{2/3} \epsilon^{-2/3} = \Omega(m_1 m_2)$, and we can switch to the trivial $O(m_1 m_2)$-time algorihm which works just by computing $a_j + b_{i-j}$ for all i, j, and taking the appropriate minima. Thus, the total time is $O\left(\frac{m_1^{1/3} m_2 L^{2/3}}{\epsilon^{2/3}}\right)$.

For the binary-tree scheme, $L = O(\log n)$, and for the NNL scheme $L = O(1)$. Thus, for constant ϵ, the running time is close to $m_1^{1/3} m_2$, greatly improving the $O(m_1 m_2)$ time of the naive algorithm. We conclude that,

Theorem 7. *For any $\epsilon > 0$,* MinSum *can be solved in $O\left(\frac{m_1^{1/3} m_2 L^{2/3}}{\epsilon^{2/3}}\right)$ deterministic time.*

There is an alternative algorithm when $\epsilon = 0$ described in [13] (which is essentially a direct application of a technique given in [8]) that uses ideas from computational geometry. We defer the description to the full version of this paper [28], but state the main theorem:

Theorem 8. *[13] If $\epsilon = 0$,* MinSum *can be solved in deterministic time*

$$O\left(\frac{m_1 m_2}{\log m_2} + m_1 + m_2\right).$$

Acknowledgment: The second author would like to thank Andrew Yao and Tsinghua University for their support while conducting part of this research.

References

[1] M. Abdalla, Y. Shavitt, and A. Wool. Key Management for Restricted Multicast Using Broadcast Encryption. In *ACM Trans. on Networking*, vol. 8, no. 4. 2000.
[2] W. Aiello, S. Lodha, and R. Ostrovsky. Fast digital identity revocation. In *Proc. of Crypto 1998.*
[3] J. Anzai, N. Matsuzaki and T. Matsumoto. A Quick Group Key Distribution Scheme with Entity Revocation. In *Proc. of Asiacrypt 1999*, LNCS 1716.
[4] T. Asano. A Revocation Scheme with Minimal Storage at Receivers. In *Proc. Of Asiacrypt 2002*, LNCS 2501, pages 433–450. Springer-Verlag, 2002.
[5] K. Azuma. Weighted Sums of Certain Dependent Random Variables. Tôhoku Math. Journ. **19** (1967) pp. 357-367.
[6] S. Berkovits. How to Broadcast a Secret. In *Proc. of Eurocrypt 1991*, LNCS 547, pages 535–541. Springer-Verlag, 1991.

[7] R. Canetti, T. Malkin and K. Nissim. Efficient Communication-Storage Tradeoffs for Multicast Encryption. In *Proc. of Eurocrypt 1999*, Springer-Verlag.
[8] T. M. Chan. *All-pairs shortest paths with real weights in $O(n^3/\log n)$ time.* In *Proc 9th WADS*, LNCS 3608, pp. 318-324, 2005.
[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to Algorithms, 1990, The MIT Press/McGraw-Hill.
[10] *Content Protection for Pre-recorded Media Specification* and *Content Protection for Recordable Media Specification*, available from `http://www.4centity.com/tech/cprm`.
[11] P. van Emde Boas. *Preserving order in a forest in less than logarithmic time and linear space.* Inform. Process. Lett. **6**(3) 1977, 80-82.
[12] P. van Emde Boas, R. Kaas, and E. Zijlstra. *Design and implementation of an efficient priority queue.* Math Systems Theory **10**(2) 1976/1977. Also, FOCS 1975.
[13] J. Erickson. Blog. Post at
http://3dpancakes.typepad.com/ernie/2005/08/chans_technique.html
[14] A. Fiat and M. Naor. Broadcast Encryption. In *Proc. of Crypto 1993*, LNCS 773, pages 480–491. Springer-Verlag, 1994.
[15] C. Gentry and Z. Ramzan. RSA Accumulator Based Broadcast Encryption. In *Proc. Information Security Conference,* 2004.
[16] Y. Han. *Deterministic sorting in $O(n \log\log n)$ time and linear space.* Journal of Algorithms, 2004, pp. 96-105.
[17] D. Halevy and A. Shamir. The LSD Broadcast Encryption Scheme. In *Proc. of Crypto 2002*, LNCS 2442, pages 47–60. Springer-Verlag, 2002.
[18] Y. Han and M. Thorup. *Sorting in $O(n\sqrt{\log\log n})$ expected time and linear space.* FOCS, 2002, pp. 135-144.
[19] Y. Kim, A. Perrig and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. In *Proc. Of ACM CCS*, 2000.
[20] R. Kumar, S. Rajagopalan and A. Sahai. Coding Constructions for Blacklisting Problems. In *Proc. of Crypto 1999*, LNCS 1666, pages 609–623. Springer-Verlag.
[21] M. Luby and J. Staddon. Combinatorial Bounds for Broadcast Encryption. In *Proc. of Eurocrypt 1998*, LNCS 1403, pages 512–526. Springer-Verlag, 1998.
[22] N. Matsuzaki, J. Anzai and T. Matsumoto. Light Weight Broadcast Exclusion Using Secret Sharing. In *Proc. of ACISP 2000*, LNCS 1841, Springer-Verlag.
[23] S. Mitra. Iolus: A Framework for Scalable Secure Multicasting. In *Proc. of ACM SIGCOMM*, September 1997.
[24] D.A. McGrew and A.T.Sherman. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. Manuscript available at `http://www.csee.umbc.edu/ ~Sherman/Papers/itse.ps`.
[25] D. Naor, M. Naor and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In *Proc. Of Crypto 2001.* Full version: ECCC Report No. 43, 2002.
[26] A. Panconesi and A. Srinivasan. *Randomized Distributed Edge Coloring via an Extension of the Chernoff-Hoeffding Bounds*, SICOMP, 1997.
[27] R. Poovendran and J.S. Baras. An Information Theoretic Analysis of Rooted-Tree Based Secure Multicast Key Distribution Schemes. In *Proc. of Crypto 1999.*
[28] Z. Ramzan and D. P. Woodruff. Fast Algorithms for the Free Riders Problem in Broadcast Encryption. IACR ePrint Archive, http://eprint.iacr.org, 2006.
[29] D.M. Wallner, E.J. Harder, and R.C. Agee. Key Management for Multicast: Issues and Architectures. Internet Draft, September 1998.
[30] C.K. Wong, M. Gouda and S.S. Lam. Secure Group Communications Using Key Graphs. In *Proc. of SIGCOMM 1998*, 1998.