Analysis and Trade-Offs for the (Complete Tree) Layered Subset Difference Broadcast Encryption Scheme

Sanjay Bhattacherjee and Palash Sarkar
Applied Statistics Unit
Indian Statistical Institute
203, B.T.Road, Kolkata, India - 700108.
{sanjayb_r,palash}@isical.ac.in

Abstract

Broadcast Encryption (BE) is an important technique for digital rights management (DRM). Two key parameters of a BE scheme are the average size of the transmission overhead and the size of the secret information to be stored by the users. The most important BE scheme till date is the subset difference (SD) scheme proposed by Naor, Naor and Lotspiech in 2001 which achieves a good balance of these two parameters. A year later, Halevy and Shamir proposed a variant of the SD scheme called the layered SD (LSD) scheme which allowed to reduce the size of user storage at the cost of increasing the transmission overhead. Since then, there has been no further study of other possible trade-offs between transmission overhead and user storage that can be obtained from the SD scheme.

In this work, we introduce several simple ideas to obtain new layering strategies with different trade-offs between user storage and transmission overhead. At one end, we introduce the notion of storage minimal layering, show that the Halevy-Shamir layering does not achieve minimal storage and describe a dynamic programming algorithm to compute layering strategies for which user storage is guaranteed to be the minimum possible. This results in user storage being 18% to 24% lower than that required by Halevy-Shamir layering schemes. At the other end, we consider the constrained minimization problem and show how to obtain BE schemes whose transmission overhead is not much more than that of the SD scheme but, whose user storage is still significantly lower than that of the SD scheme.

The original SD and the LSD algorithms are defined only when the number of users is a power of two. In an earlier work, we have shown how to handle arbitrary number of users in the SD scheme. Here this is extended to the LSD scheme. Finally, we obtain an $O(r \log^2 n)$ time algorithm to compute the average transmission overhead in any layering based SD scheme with n users out of which r are revoked.

Keywords: Broadcast encryption; subset difference; layering; transmission overhead; user storage

1 Introduction

Many real-life scenarios can be modelled as follows. There is a set of users and a centre which broadcasts messages. For each message, the centre decides on a set of users which should be able to access the message while the other users should not be able do so. The two subsets are respectively called *privileged* and *revoked* and they form a partition of the set of all users. A cryptographic system achieving such a functionality is called a broadcast encryption scheme [Ber91, FN93]. Practical applications include Pay-TV and more generally digital rights management.

A BE scheme can be based on symmetric key cryptography or on public key cryptography. In a symmetric key based BE scheme, the centre pre-distributes secret information to the users during a set-up phase. For a transmission, a session key is generated and the message is encrypted using the session key. Next the session key is encrypted using several other keys which are determined by the set of privileged users. The additional encryptions of the session key constitute the *header* while the actual encryption of the message is called the *body*. To decrypt, a privileged user can use its secret information to obtain one of the keys with which the session key has been encrypted. Decrypting the appropriate component of the header with this key yields the session key and then decrypting the body with the session key yields the message. The two important parameters of a BE scheme are the length of the header (as given by the number of encryptions of the session key) and the size of secret information to be stored by a user. It is desirable to decrease both as far as possible, but, in most schemes it turns out that decreasing one increases the other.

BE based on public key cryptography allows users to have public and private keys. There is no centre and anybody can encrypt to a subset of users. We do not consider public key BE in this work and instead refer the reader to relevant work such as [BF99, DF03, BGW05]. For this paper, by BE we will mean symmetric key BE.

Naor, Naor and Lotspiech (NNL) [NNL01] introduced an important BE scheme called the subset difference (SD) method. This scheme has been adopted as a standard for content protection in HD-DVD and Blu-ray discs [AAC]. The NNL-BE scheme is defined for n users where n is a power of two, i.e., $n = 2^{\ell_0}$ for some $\ell_0 \geq 0$. The users are considered to be the leaves of a full binary tree having ℓ_0 levels. Each user needs to store $\ell_0(\ell_0 + 1)/2$ k-bit strings where k is the key length of the underlying symmetric key cryptosystem. If r users are revoked, then the worst case header length (i.e., the number of encryptions of the session key) is $2r - 1$ [NNL01], while the average case header length turns out to be at most $1.25r$ for practical situations (see [BSar] for a detailed analysis). The trade-off between user storage and average header length turns out to be very well suited for real-life applications. Further, the scheme itself is quite elegant and reasonably easy to implement.

A later work by Halevy and Shamir [HS02] introduced a variant of the SD method called the layered subset difference (LSD) scheme. This is also defined for n users where $n = 2^{\ell_0}$. The basic idea is to partition the tree into several layers which gives the name of the scheme. A different trade-off is obtained. User storage is reduced in the LSD method to $\ell_0^{3/2} - \ell_0$ but, the maximum possible header length grows to $4r - 3$. In [HS02], based on simulation results, it is remarked that the average header length is around $2r$. Compared to the SD method, the LSD method reduces the user storage at the cost of increasing the header length.

Our Contributions: We make a detailed analysis of the idea of layering introduced in [HS02]. It is shown that the layering strategy introduced by Halevy and Shamir does not minimize the user storage. This is true for concrete values of n and asymptotically speaking the difference in the user storage between the HS layering strategy and a minimal storage layering strategy goes to infinity. In the HS layering strategy the root node of the user tree is treated as a special level. We show that by simply removing this condition yields a scheme where the user storage is $\ell_0^{3/2} - \ell_0 - (\ell_0 - \ell_0^{1/2})$ and has a negligible effect on the average header length. The notion of storage minimal layering is introduced. For such a layering strategy, user storage requirement is the minimum possible. An $O(\ell_0^3)$ time and $O(\ell_0^2)$ space dynamic programming algorithm is presented to compute storage minimal layerings. The resulting user storages are between 18% to 24% lower than that required by the Halevy-Shamir layering scheme.

Simply minimizing user storage is only one aspect of the problem. We consider the constrained minimization problem whereby one tries to minimize the user storage but, without increasing the average header length significantly beyond that achieved by the SD scheme. This is a difficult problem to solve analytically. Instead, we show how to tackle the problem empirically. Given some idea about the number of users that would be revoked, we show how one may use this information to design a layering strategy for which the average header length is almost as small as the SD scheme. The user storage for such a layering scheme is significantly less than that of the SD scheme. In fact, they are comparable to the LSD scheme storage requirements. Concrete practical examples are provided.

Having the number of users to be a power of two can be restrictive for some applications. For the SD scheme, this restriction has been removed in [BSar] by using the notion of complete binary trees. The resulting scheme has been called the complete tree SD (CTSD) scheme. Here, we extend the ideas from [BSar] to the LSD scheme to obtain the complete tree LSD (CTLSD) scheme. For the CTSD scheme, an $O(r \log n)$ time algorithm to compute the average header length has been obtained in [BSar]. Following this approach, here we obtain an $O(r \log^2 n)$ algorithm to compute the average header length for the CTLSD scheme. Our study of the average header length of the CTLSD method as well as all experimental results have been computed using this algorithm.

Previous and Related Works: Subsequent to [NNL01, HS02], there have been some follow-up work analyzing the average header lengths of the SD and LSD schemes. In [PB06], a generating function is obtained for counting the number of ways p users out of n can be given access privilege so that the header length will be h. For a given n and p, the generating function was used to obtain equations to compute the expected header length. The authors however mentioned that their equations were "complex to compute and difficult to gain insight from". Consequently they went forward to find *approximations* for the same. In contrast, our algorithm is efficient and simpler to implement. In [EOPR08] this analysis of the expected header length was continued and it was shown that the standard deviations are small compared to the expected values, as the number of users gets large. Other combinatorial studies of the SD method has been done in [MMW09, AK08]. All of these works considered the number of users to be a power of two. In [BSar], this condition was relaxed and the SD method was extended to the CTSD method. A detailed combinatorial and probabilistic analysis of the CTSD method was carried out.

Several works [LS98, PGM04] on the combinatorics behind broadcast encryption schemes and different generic bounds on the efficiency parameters have been done. In [AKI03], a generic method for constructing BE schemes from pseudo-random generators was proposed. There have been several other works focussing on different aspects of BE [Sti97, SW98, GST04, AK08].

Several other BE schemes have been proposed. Linear algebraic techniques have been used in [PGMM03] to find a family of broadcast encryption schemes called linear broadcast encryption schemes. The same authors had also proposed key pre-distribution techniques based on linear algebraic techniques in [PGMM02]. Another interesting work on BE is [JHC$^+$05], that works on the idea of "one key per punctured interval". In [JHC$^+$05], the worst case header length has been brought down to r or below for the first time, but at the cost of increasing user storage. But, the method is more complicated than the SD scheme and the user storage requirement is rather high.

A related and required aspect of BE schemes is *traitor tracing* [CFN94, FT01, NP98, KY01, SSW01], where pirate decryption boxes are used to trace the compromised keys. The traitor tracing method for the SD scheme can be modified to obtain a traitor tracing method for the CTLSD scheme and so we do not discuss this issue here.

2 Subset Cover Framework

Suppose there are n users. In the subset cover revocation framework, a collection $\mathcal{S}$ of subsets of $\{1, \ldots, n\}$ is defined in a manner such that any set $S \in \mathcal{S}$ has an associated key and any subset of $\{1, \ldots, n\}$ which is not in $\mathcal{S}$ does not have any key associated with it. For a user u, let $\mathcal{S}_u = \{S \in \mathcal{S} : u \in S\}$. User u is given secret information I_u such that it can construct the key associated with any set in $\mathcal{S}_u$.

During the actual broadcast, some users are revoked and some are privileged. Suppose that a subset T of the users are privileged. A cover finding algorithm determines a collection of pairwise disjoint subsets of $\mathcal{S}$ whose union is T. This collection of subsets is called the subset cover. The actual message is encrypted with a session key and the session key is encrypted with the keys associated with the subsets in the cover. The encrypted message forms the body while the different encryptions of the session key forms the header. So, the number of subsets in the cover determine the header length of the broadcast. Loosely speaking, this number itself is called the header length of the transmission.

To decrypt, a user first determines to which subset of the cover it belongs. Then, using its secret information, it generates the key associated with this subset. Decrypting the appropriate component of the header with this key, the user obtains the session key and then decrypting with the session key the user obtains the actual message.

Two parameters are of crucial interest. The size of the secret information I_u that is to be stored by a user u and the average or expected length of a broadcast header. Basic intuition tells us that as the number of elements in $\mathcal{S}$ grows, it should be possible to cover a privileged set T with lesser number of elements and so the average header length will decrease. On the other hand, as $\mathcal{S}$ grows, the size of $\mathcal{S}_u$

also grows and this should lead to an increase in the size of I_u. Thus, the average header length and the user storage are two competing parameters.

The Subset Difference Scheme. The SD scheme introduces a major novelty in defining $\mathcal{S}$ such that there is a compact way of representing I_u. In the original SD scheme, the number of users n is a power of 2, say $n = 2^{\ell_0}$. Consider the users to be the leaves of a full binary tree. Each node in the tree represents the users at the leaf level of the tree rooted at that node. Suppose i is a node of the tree and let $\mathcal{S}^{(i)}$ denote the leaves of the subtree rooted at i. Let j be a node in the subtree rooted at i. Then for the SD scheme, the set $\mathcal{S}$ consists of the subsets $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$ for all possible choices of node i and all possible nodes $j \neq i$ in the subtree rooted at i.

A clever algorithm is used to define the key associated with a set $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$. First each node i in the tree is assigned an independent and uniform random label $LABEL_i$. A cryptographically strong pseudo-random generator (PRG) $G : \{0,1\}^k \rightarrow \{0,1\}^{3k}$ is used. Let $G(seed)$ be written as the concatenation of 3 k-bit strings $G_L(seed)$, $G_M(seed)$ and $G_R(seed)$. Suppose that a node j in the subtree rooted at node i is reached from node i by the moves 'left', 'left' and 'right'. Then the label of j derived from $LABEL_i$ is $LABEL_{i,j} = G_R(G_L(G_L(LABEL_i)))$ and the key associated with the set $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$ is $L_{i,j} = G_M(G_R(G_L(G_L(LABEL_i))))$. This easily extends to any appropriate pair of nodes i and j. The string $L_{i,j}$ is a k-bit string and the value of k is determined by the key size of the underlying encryption algorithm.

Recall that users are at the leaf level of the tree. The leaf level is numbered 0 and level numbers increase up to ℓ_0 which is the level number of the root. For any user u, the user storage I_u is defined in the following manner. Consider the path from the node u to the root and let i be a node on this path at level $\ell > 0$ of the tree. Let $i_1, \ldots, i_\ell$ be the siblings of the nodes on the path from u to i (including u but not including i). Then for each such i, user u gets the labels $LABEL_{i,i_1}, LABEL_{i,i_2}, \ldots, LABEL_{i,i_\ell}$. The value of ℓ varies from 0 to ℓ_0 and so each user gets $\ell_0(\ell_0 + 1)/2$ labels. The total size of I_u is $k\ell_0(\ell_0 + 1)/2$ bits where k is the size of the seed of the PRG. Since k is fixed, it is enough to consider only the number of labels as determining the size of user storage.

The labels provided to a user is sufficient for the user to construct the key corresponding to any element in $\mathcal{S}_u$. To see this suppose that i is a node on the path from u to the root and j is a node in the subtree rooted at i such that $u \in \mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$. Since u is not in $\mathcal{S}^{(j)}$ and both u and j are in the subtree rooted at i, the paths to root from these two nodes intersect for the first time at some node v which is also in the subtree rooted at i. Let v_1 be the first node in the path from v to j. Then v_1 is the sibling of some node v_2 in the path from u to i and so u has $LABEL_{i,v_1}$. From this label, u can generate $LABEL_{i,j}$ by applying G_L and G_R appropriately and so can generate $L_{i,j} = G_M(LABEL_{i,j})$. This $L_{i,j}$ is the key corresponding to the set $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$. So, u can generate keys for any subset in $\mathcal{S}_u$.

It is also required to argue that u cannot generate keys for any other subset in $\mathcal{S}$. In the SD scheme, any subset in $\mathcal{S}$ is of the form $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$. If u is not in such a subset, then u is either not in $\mathcal{S}^{(i)}$ or it is in $\mathcal{S}^{(j)}$. In either case, it is not too difficult to see that u does not obtain information which allows it to generate $L_{i,j}$. See [NNL01] for more details.

The Layered Subset Difference Scheme. The point of the LSD scheme is to reduce the user storage in the SD scheme at the cost of increasing the header length. Reduction in the user storage is achieved by reducing the size of $\mathcal{S}$. As in the SD scheme, the LSD scheme also considers the number of users to be of the form 2^{ℓ_0} where the users form the leaves of a full binary tree. The major difference between the SD and the LSD schemes is that in the LSD scheme the levels of the tree are partitioned into layers. Levels which form the boundary of the layers are called special levels.

A subset $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$ is defined to be in $\mathcal{S}$ if either of the following two conditions hold:

- node i is at a special level;
- or, node i is not at a special level but, node j is in the same layer as level i.

This reduces the size of $\mathcal{S}$ and consequently of $\mathcal{S}_u$. As a result, the size of I_u also reduces as we explain below. The distribution of labels is done as follows. Suppose that u is a user (i.e., a leaf node) and

i is a node at level ℓ in the path from u to the root and $i_1, \ldots, i_\ell$ are the siblings of the nodes in the path from u to i. If ℓ is a special level, then u is given $LABEL_{i,i_1}, \ldots, LABEL_{i,i_\ell}$ as in the SD scheme. Suppose ℓ is not a special level. Let ℓ' be the first special level below i and consider the segment of the path from u to i which lies between ℓ' and ℓ. Suppose $i_m, \ldots, i_\ell$ are the siblings of the nodes on this segment. Then u gets $LABEL_{i,i_m}, \ldots, LABEL_{i,i_\ell}$. The net effect is that if i is not at a special level, it generates labels only up to the next special level. This leads to the reduction in the user storage.

The reduction in user storage is achieved at an increase in the size of the header length. Suppose i is not at a special level and j is in the sub-tree rooted at i but not in the same layer as i. The SD scheme would associate the set $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$ to such an (i,j) pair. In the LSD scheme, this set is not present. Instead, the header computation algorithm will cover this set in the following manner. Let k be the node in the first special level as one moves down the path from i to j. The sets $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(k)}$ and $\mathcal{S}^{(k)} \setminus \mathcal{S}^{(j)}$ are both present in the LSD scheme and it is easy to see that

$$\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)} = \left(\mathcal{S}^{(i)} \setminus \mathcal{S}^{(k)}\right) \bigcup \left(\mathcal{S}^{(k)} \setminus \mathcal{S}^{(j)}\right).$$

This can be viewed as a two-way split of the set $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$. The work [HS02] also consider the possibility of multi-way split. But, the authors conclude that this leads to further reduction in user storage only for impractical values of the number of users. In this paper, we will not consider multi-way split.

Halevy-Shamir (HS) Layering: In [HS02], the special layers are identified in the following manner. Let $d \leq \ell_0$ be a fixed positive integer and write $\ell_0 = d(e-1) + p$ where $1 \leq p \leq d$. Then the special levels are

$$\ell_0,\ \ell_0 - d,\ \ell_0 - 2d,\ \ldots,\ \ell - d(e-1),\ 0.$$

So, there are a total of $e+1$ special levels and e layers out of which $e-1$ layers are of length d each and the last layer is of length p. Note that p can equal d which will lead to e layers each of length d. (If $p < d$, then [HS02] does not consider the bottom-most level 0 to be a special level; notationally, we find it more convenient to always have level 0 as a special level and this does not have any effect on either the user storage or the header length.)

In [HS02], the suggested value of d is $\lceil \sqrt{\ell_0} \rceil$. By the Halevy-Shamir (HS) layering strategy we will mean the strategy with layer lengths $d, d, \ldots, d, p$ with $d = \lceil \sqrt{\ell_0} \rceil$.

3 General Layering Strategy

We analyze the LSD scheme where the layer lengths are not necessarily equal. Suppose there are $(e+1)$ special levels and the numbers of the special levels are $\ell_0 > \ell_1 > \ldots > \ell_{e-1} > \ell_e = 0$. Let $\boldsymbol{\ell} = (\ell_0, \ldots, \ell_e)$.

The user storage for such a strategy can be calculated as follows. Corresponding to each special level ℓ_i, a user has to store ℓ_i labels. Now consider the nodes in the layer bordered by ℓ_i and ℓ_{i+1}. Corresponding to any non-special level j in this layer a user has to store $j - \ell_{i+1}$ labels. So, the total number of labels that is required to be stored by a user considering both special and non-special levels is given by the following formula.

$$\begin{aligned}
\mathsf{storage}_0(\boldsymbol{\ell}) &= \sum_{i=0}^{e-1} \ell_i + \sum_{i=0}^{e-1} \sum_{j=\ell_{i+1}+1}^{\ell_i - 1} (j - \ell_{i+1}) \\
&= \sum_{i=0}^{e-1} \ell_i + \sum_{i=0}^{e-1} \sum_{j=1}^{\ell_i - \ell_{i+1} - 1} j \\
&= \sum_{i=0}^{e-1} \ell_i + \frac{1}{2} \sum_{i=0}^{e-1} (\ell_i - \ell_{i+1})(\ell_i - \ell_{i+1} - 1). \qquad (1)
\end{aligned}$$

It is sometimes more convenient to use another formulation. For $1 \leq i \leq e$, define $d_i = \ell_{i-1} - \ell_i$ so that d_i's are positive integers whose sum is ℓ_0. Conversely, given any sequence of positive integers

$\mathbf{d} = (d_1, \ldots, d_e)$ whose sum is ℓ_0, it is possible to define a layering scheme where $\ell_i = \ell_0 - \sum_{j=1}^{i} d_j$. Using (1) we have the following.

$$\mathsf{storage}_0(\boldsymbol{\ell}) \;=\; \ell_0(e+1) - \sum_{i=1}^{e}(e-i+1)d_i + \frac{1}{2}\sum_{i=1}^{e} d_i(d_i - 1). \qquad (2)$$

Some consequences of (1) and (2) are mentioned below.

1. If all the d_i's are equal to d and $\ell_0 = e \times d$, then $\mathsf{storage}_0(\boldsymbol{\ell})$ is given by $\ell_0(e+d)/2 - \ell_0$. This shows that the user storage using e layers of length d each is the same as the user storage using d layers of length e each. If all the layer lengths are equal, then the problem of minimizing the user storage is that of minimizing the sum $e + d$ subject to the constraint $ed = \ell_0$. From this it is easy to see that the minimum value is attained for $e = d = \sqrt{\ell_0}$ and the corresponding value of user storage is $\ell_0^{3/2} - \ell_0$. Since, ℓ_0 may not be a perfect square one may take d to be $\lceil\sqrt{\ell_0}\rceil$. This justifies the choice made in [HS02]. Note that the minimization here is in the context of all the layer lengths being equal.

2. If each d_i equals 1, then each level is a special level and the resulting scheme becomes identical to the SD scheme. Similarly, if $e = 1$, i.e., only the root and the leaf levels are the special levels, then also the resulting scheme is the SD scheme. In both the cases the expression given by (2) is maximized and the value is $\ell_0(\ell_0 + 1)/2$.

3. Let $\mathbf{d} = (d_1, \ldots, d_e)$, $\mathbf{d}' = (d_1, \ldots, d_{e-1}, \underbrace{1, \ldots, 1}_{d_e})$ and suppose $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ are the corresponding layering strategies. Then the labels stored by a user u are the same for both cases and consequently, $\mathsf{storage}_0(\boldsymbol{\ell}) = \mathsf{storage}_0(\boldsymbol{\ell}')$. In other words, at the end, having a single layer of length d_e is the same as having d_e layers of length 1 each.

4. Suppose $\mathbf{d} = (d_1, \ldots, d_e)$ with $d_1 \geq d_2 \geq \cdots \geq d_e$ and $\mathbf{d}' = (d_{\pi(1)}, \ldots, d_{\pi(e)})$ where π is a permutation of $\{1, \ldots, e\}$. Let $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ be the corresponding sequences of special levels. Then $\mathsf{storage}_0(\boldsymbol{\ell}) \leq \mathsf{storage}_0(\boldsymbol{\ell}')$. This can be seen by noting that the quantity $\ell_0(e+1)$ and the quadratic terms in (2) are the same in both cases. A simple argument then shows the required inequality. As an example, suppose $\ell_0 = 12$ and fix $e = 8$. Then the scheme having $(d_1, d_2, \ldots, d_8) = (2, 2, 2, 2, 1, 1, 1, 1)$ requires a storage of 50 labels whereas the scheme having $(d_1, d_2, \ldots, d_8) = (1, 1, 1, 1, 2, 2, 2, 2)$ requires a storage of 66 labels.

5. Consider the HS layering strategy where $\ell_0 = d(e-1) + p$ with $1 \leq p \leq d$ and $d = \lceil\sqrt{\ell_0}\rceil$. Then $\mathbf{d} = (\underbrace{d, \ldots, d}_{e-1}, p)$. Since p can be much less than d one can consider a strategy where the layer lengths are balanced. Write $\ell_0 = d(e-1) + p = (e-d+p)d + (d-p)(d-1)$ and define $\mathbf{d}' = (\underbrace{d, \ldots, d}_{e-d+p}, \underbrace{d-1, \ldots, d-1}_{d-p})$. Let $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ be the corresponding sequences of special levels. Then, one can show that $\mathsf{storage}_0(\boldsymbol{\ell}) = \mathsf{storage}_0(\boldsymbol{\ell}')$. Experimental results show that the average header lengths for both strategies are similar with that corresponding to $\boldsymbol{\ell}'$ being slightly smaller. As an example, for $\ell_0 = 18$, $\mathbf{d}' = (5, 5, 4, 4)$ yields lesser expected header lengths than $\mathbf{d} = (5, 5, 5, 3)$ for all r between 256 and 16384 while the user storage 75 is the same for both. These expected header lengths are computed using the algorithm (described later) for computing average header length.

6. Let $\mathbf{d} = (d_1, \ldots, d_e)$ and suppose that $d_i = d + \delta$ and $d_{e-j+1} = d$, i.e., the i-th entry from the front is $d + \delta$ and the j-th entry from the end is d. Suppose that $\mathbf{d}'$ is obtained from $\mathbf{d}$ by incrementing d_i (i.e., changing its value to $d + \delta + 1$) and decrementing d_{e-j+1} (i.e., changing its value to $d - 1$). Let $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ be the corresponding sequences of special levels. A simple calculation based on (2) shows that $\mathsf{storage}_0(\boldsymbol{\ell}) - \mathsf{storage}_0(\boldsymbol{\ell}') = (e - i - j - \delta)$. So, if $e > i + j + \delta$, then it is possible to reduce storage by incrementing d_i and decrementing d_{e-j+1}. This simple observation can be used to show that the HS layering strategy does not minimize the storage requirement. Let $d = \lceil\sqrt{\ell_0}\rceil$

and assume that d divides ℓ_0 such that $\ell_0 = d \times e$. (If d does not divide ℓ_0, then it is easy to modify the argument below.) Then the HS layering scheme will be $\mathbf{d} = (d, d, \ldots, d)$. Let $\theta \geq 1$ be such that $e > 2\theta$ and define

$$\mathbf{d}' = (\underbrace{d+1, \ldots, d+1}_{\theta}, d, \ldots, d, \underbrace{d-1, \ldots, d-1}_{\theta}).$$

Then $\mathsf{storage}_0(\boldsymbol{\ell}) = \mathsf{storage}_0(\boldsymbol{\ell}') + \theta(e-\theta-1)$. The gap $\theta(e-\theta-1)$ is positive. In fact, as ℓ_0 goes to infinity, e also goes to infinity and the gap goes to infinity. This, however, does not imply that the layering strategy from $\mathbf{d}'$ provides the minimum storage. There could be other more non-uniform layering strategies which further reduce the storage requirement.

Root as a non-special level: In [HS02], the root level ℓ_0 is always taken as a special level. It is possible to obtain further reduction in user storage if we allow the root level to be a non-special level. Having the root as a special level contributes ℓ_0 labels to the user storage. If instead the root is made a non-special level, then its contribution to the user storage will be $\ell_0 - \ell_1$ labels. Given a sequence of level numbers $\boldsymbol{\ell}$, let $\mathsf{storage}_1(\boldsymbol{\ell})$ be the number of labels required to be stored when the root is not a special level (and so, ℓ_1 is the first special level). Then the following relation holds.

$$\mathsf{storage}_1(\boldsymbol{\ell}) \quad = \quad \mathsf{storage}_0(\boldsymbol{\ell}) - \ell_1. \tag{3}$$

Equivalently,

$$\mathsf{storage}_1(\ell_0, \ldots, \ell_e) \quad = \quad \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 + 1)}{2} + \mathsf{storage}_0(\ell_1, \ldots, \ell_e). \tag{4}$$

So, not having the root to be a special level reduces the storage requirement by ℓ_1 labels. This can be quite significant. Consider the HS layering strategy where $\ell_0 = d \times e$ and so $\boldsymbol{\ell} = (\ell_0, \ell_1, \ldots, \ell_e)$ where $\ell_i - \ell_{i+1} = d$ for $0 \leq i < e$. In this case, $\mathsf{storage}_0(\boldsymbol{\ell}) = \ell_0^{3/2} - \ell_0$ and $\mathsf{storage}_1(\boldsymbol{\ell}) = \ell_0^{3/2} - \ell_0 - (\ell_0 - \ell_0^{1/2})$.

Given a layering strategy $\boldsymbol{\ell} = (\ell_0, \ldots, \ell_e)$, there are $e+1$ special levels irrespective of whether the root is a special level or not. If the root is a special level, then there are e layers and if the root is not a special level, then there are $e+1$ layers with the top layer having only one special level at the bottom.

It is important to understand the effect on the header length when the root level is not special. During the computation of the cover, suppose that the root generates an SD subset, i.e., the SD cover finding algorithm returns a subset of the form $\mathcal{S}^{(0)} \setminus \mathcal{S}^{(j)}$. Since the root is not at a special level, this subset may be split into two if j is not in the first layer. We argue that for reasonable values of r (the number of revoked users), this effect is negligible. In fact, the argument is that the probability of the root generating an SD subset itself is small.

The root generates an SD subset only if exactly one of the two subtrees of the root node contains all the revoked users. Suppose the revoked users are uniformly distributed, i.e., r users are uniformly sampled one-by-one without replacement and revoked. Then the probability that the left subtree does not have any revoked user (and consequently the right subtree contains all of them) is

$$\begin{aligned}
&\left(1 - \frac{n/2}{n}\right)\left(1 - \frac{n/2}{n-1}\right)\cdots\left(1 - \frac{n/2}{n-r+1}\right) \\
&= \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{2\left(1-\frac{1}{n}\right)}\right)\left(1 - \frac{1}{2\left(1-\frac{2}{n}\right)}\right)\cdots\left(1 - \frac{1}{2\left(1-\frac{r-1}{n}\right)}\right)
\end{aligned}$$

The probability that the right subtree does not have any revoked user is also equal to this value. So, the total probability that the header generates a subset is twice this value. For fixed n as r increases, this probability becomes vanishingly small. Even for values of r as low as 20 or so, this probability is almost negligible. Hence, there will be almost no effect on the header length if the root level is not made special.

3.1 Storage Minimal Layering

For a given value of ℓ_0, let $\mathrm{SML}_0(\ell_0)$ denote a layering strategy $\boldsymbol{\ell}$ (or equivalently is given by the sequence of differences $\mathbf{d}$) including the root level, such that $\mathsf{storage}_0(\boldsymbol{\ell})$ takes the minimum value among all possible layering strategies for a tree with ℓ_0 levels. Let $\#\mathrm{SML}_0(\ell_0)$ denote $\mathsf{storage}_0(\boldsymbol{\ell})$ where $\boldsymbol{\ell}$ is a storage minimal layering strategy. Similarly define $\mathrm{SML}_1(\ell_0)$ and $\#\mathrm{SML}_1(\ell_0)$ that exclude the root level from being special.

We describe a dynamic programming based algorithm to compute $\mathrm{SML}_0(\ell_0)$ (and subsequently $\mathrm{SML}_1(\ell_0)$). The idea of the algorithm is explained as follows. For a fixed value of ℓ_0, the number of layers e can vary from 1 to ℓ_0. The cases $e = 1$ and $e = \ell_0$ correspond to the SD scheme and in these two cases the user storage is known to be equal to $\ell_0(\ell_0 + 1)/2$. Let $\mathrm{SML}_0(e, \ell_0)$ denote a storage minimal layering *using exactly* e *layers*. Clearly, the following relation holds.

$$\#\mathrm{SML}_0(\ell_0) \;=\; \min_{1\le e\le \ell_0} \#\mathrm{SML}_0(e, \ell_0). \tag{5}$$

Also,

$$\#\mathrm{SML}_0(e, \ell_0) \;=\; \min_{(\ell_0,\ldots,\ell_e)} \mathsf{storage}_0(\ell_0, \ell_1, \ldots, \ell_e) \tag{6}$$

where the minimum is over all possible layering strategies $(\ell_0, \ell_1, \ldots, \ell_e)$. The following relation is obtained from (1).

$$\begin{aligned}
\mathsf{storage}_0(\ell_0, \ell_1, \ldots, \ell_e) &= \ell_0 + \ell_1 + \cdots + \ell_e + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} \\
&\quad + \frac{(\ell_1 - \ell_2)(\ell_1 - \ell_2 - 1)}{2} + \cdots + \frac{(\ell_{e-1} - \ell_e)(\ell_{e-1} - \ell_e - 1)}{2} \\
&= \ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} + \mathsf{storage}_0(\ell_1, \ldots, \ell_e).
\end{aligned} \tag{7}$$

Consequently,

$$\#\mathrm{SML}_0(e, \ell_0) \;=\; \min_{1\le \ell_1 < \ell_0} \left(\ell_0 + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 - 1)}{2} + \#\mathrm{SML}_0(e-1, \ell_1) \right). \tag{8}$$

This relation is the basis for the algorithm. Let Tab be an $\ell_0 \times \ell_0$ table such that $\mathsf{Tab}[e][\ell_0] = \#\mathrm{SML}_0(e, \ell_0)$. A simple $O(\ell_0^3)$ time dynamic programming algorithm can fill up this table as follows.

```
for $\ell = 1, 2, \ldots, \ell_0$,
   $\mathsf{Tab}[1][\ell] = \mathsf{Tab}[\ell][\ell] = \ell(\ell + 1)/2$;
for $\ell = 1, 2, \ldots, \ell_0$,
   for $e = 2, 3 \ldots, \ell_0 - 1$,
      $\mathsf{Tab}[e][\ell] = \min_{1\le \ell_1 < \ell} (\ell + (\ell - \ell_1)(\ell - \ell_1 - 1)/2 + \mathsf{Tab}[e-1][\ell_1])$.
```

Using (5) provides $\#\mathrm{SML}_0(\ell_0)$ as the minimum value in column number ℓ_0 of Tab. Note that the minimum may occur for more than one possible value of e. These values of ℓ_1 are reported during the computation. Let $\Lambda(e, \ell_0)$ be the list of all possible values of ℓ_1 for which (8) holds. The above method can be extended to generate all possible layering strategies for which user storage is minimized.

An SML_0 layering strategy $\boldsymbol{\ell}$ can be generated as follows. Start with $\boldsymbol{\ell}$ as the list containing only ℓ_0 and keep on appending in the following manner to obtain the complete sequence. Let e be one of the possibilities for which $\mathsf{Tab}[e][\ell_0]$ takes the minimum value; choose ℓ_1 as any one value from $\Lambda(e, \ell_0)$ and append to $\boldsymbol{\ell}$; choose ℓ_2 as any one value from $\Lambda(e-1, \ell_1)$ and append to $\boldsymbol{\ell}$; continue until 0 is appended to the list. All SML_0 strategies can be generated by looping over all possible values of e, all possible values of ℓ_1, all possible values of ℓ_2 and so on.

Once Tab is prepared, computing $\#\mathrm{SML}_1(\ell_0)$ using (4) is easy.

$$\begin{aligned}
\#\mathrm{SML}_1(\ell_0) &= \min_{e}\min_{\ell_1} \left(\#\mathrm{SML}_0(e-1, \ell_1) + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 + 1)}{2} \right) \\
&= \min_{e}\min_{\ell_1} \left(\mathsf{Tab}[e-1][\ell_1] + \frac{(\ell_0 - \ell_1)(\ell_0 - \ell_1 + 1)}{2} \right).
\end{aligned} \tag{9}$$

Table 1: Comparison of user storage and expected header lengths between HS-LSD and SML. The tuples contain header lengths normalized with the SD header lengths corresponding to the values of r in $(r_{\min}, \ldots, r_{\max})$ respectively.

ℓ_0	$r_{\min}$	$r_{\max}$	scheme	special levels	storage	normalized header lengths for $(r_{\min}, \ldots, r_{\max})$
12	2^2	2^8	SD	12, 0	78	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			HS	12, 8, 4, 0	**42**	(**1.69, 1.57**, 1.53, 1.45, 1.38, 1.31, 1.26, 1.22, 1.18, 1.15)
			SML_0	12, 8, 5, 3, 1, 0	40	(**1.68, 1.54**, 1.53, 1.49, 1.46, 1.44, 1.43, 1.42, 1.42, 1.42)
			SML_1	8, 5, 3, 1, 0	**32**	(**1.68, 1.54**, 1.53, 1.49, 1.46, 1.44, 1.43, 1.42, 1.42, 1.42)
16	2^6	2^{12}	SD	16, 0	136	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			HS	16, 12, 8, 4, 0	**64**	(**1.66, 1.57**, 1.53, 1.45, 1.38, 1.31, 1.26, 1.22, 1.18, 1.15)
			SML_0	16, 11, 7, 4, 2, 0	61	(**1.64, 1.53**, 1.53, 1.52, 1.50, 1.47, 1.44, 1.42, 1.39, 1.38)
			SML_1	11, 7, 4, 2, 0	**50**	(**1.64, 1.53**, 1.53, 1.52, 1.50, 1.47, 1.44, 1.42, 1.39, 1.38)
20	2^8	2^{14}	SD	20, 0	210	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			HS	20, 15, 10, 5, 0	**90**	(1.68, **1.64, 1.64, 1.60**, 1.55, 1.50, 1.45, 1.41, 1.38, 1.34)
			SML_0	20, 14, 9, 6, 3, 1, 0	85	(1.71, **1.56, 1.57, 1.58**, 1.58, 1.57, 1.56, 1.55, 1.54, 1.54)
			SML_1	15, 10, 6, 3, 1, 0	**70**	(1.68, 1.64, 1.65, 1.63, 1.60, 1.58, 1.56, 1.55, 1.55, 1.54)
24	2^{10}	2^{16}	SD	24, 0	300	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			HS	24, 19, 14, 9, 4, 0	**116**	(1.63, **1.68, 1.63, 1.61, 1.61, 1.61, 1.62, 1.62, 1.63, 1.64**)
			SML_0	24, 18, 12, 7, 3, 1, 0	112	(1.63, **1.65, 1.61, 1.58, 1.56, 1.56, 1.56, 1.56, 1.57, 1.57**)
			SML_1	18, 12, 8, 5, 3, 1, 0	**94**	(1.68, **1.64**, 1.65, 1.63, **1.60, 1.58, 1.57, 1.56, 1.55, 1.54**)
28	2^{10}	2^{16}	SD	28, 0	406	(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)
			HS	28, 22, 16, 10, 4, 0	**146**	(1.69, **1.70, 1.75, 1.75, 1.74, 1.73, 1.72, 1.71, 1.70, 1.69**)
			HS′	28, 22, 16, 10, 5, 0	146	(1.69, 1.70, 1.75, 1.75, 1.74, 1.73, **1.71, 1.70, 1.69, 1.68**)
			SML_0	28, 21, 15, 10, 6, 3, 1, 0	140	(1.74, **1.63, 1.67, 1.71, 1.72, 1.72, 1.71, 1.70, 1.69, 1.68**)
			SML_1	22, 16, 11, 7, 4, 2, 0	**119**	(1.69, **1.68, 1.72, 1.71, 1.69, 1.67, 1.66, 1.64, 1.63, 1.63**)

Table 2: The number of $SML_0(\ell_0)$ and $SML_1(\ell_0)$ layering strategies for various values of ℓ_0.

ℓ_0	no. of $SML_0(\ell_0)$ layerings	no. of $SML_1(\ell_0)$ layerings
12	10	10
16	6	15
20	6	1
24	35	35
28	1	8

The first minimization is over the number of layers and the second minimization is over the value of the first special level. The possible corresponding layering strategies can also be easily recovered. It is to be noted that the $SML_1(\ell_0)$ layerings are due to the minimization of the user storage by assuming the root to be at a non-special level. It can be seen from (8) and (9) that in an $SML_0(\ell_0)$ layering, if the root is made non-special, it might not necessarily result in an $SML_1(\ell_0)$ layering and vice versa.

Table 1 shows values of user storage for SML strategies for some ℓ_0. For comparison, we also show the storage requirements for the SD scheme and the HS layering strategy. Compared to the SD scheme, the HS layering strategy reduces the storage requirement very significantly. Compared to the HS scheme the value of $\#SML_0(\ell_0)$ is slightly smaller and the value of $\#SML_1(\ell_0)$ is about 18% to 24% lower for the newly suggested values of $\boldsymbol{\ell}$. So, given a value of ℓ_0, if the requirement is to minimize the user storage, then the SML strategies offer better alternatives. They also guarantee that further lowering of storage cannot be achieved by 2-way splitting of SD subsets.

The effect of $SML_0(\ell_0)$ and $SML_1(\ell_0)$ strategies on the average header length is also shown in Table 1. For computing the average header lengths, we have considered ten values of r equally spaced between $r_{\min}$ and $r_{\max}$. The reported values are the average header lengths of the different schemes normalized by the average header length of the SD scheme. As an example, the first value 1.69 corresponding to the row for HS and $\ell_0 = 28$ means that with $n = 2^{28}$ users out of which $r = 2^{10}$ are uniformly revoked, the average header length of the HS layering strategy is 1.69 times that of the SD scheme.

One may note the following points.

Table 3: List of $\text{SML}_0(\ell_0)$ and $\text{SML}_1(\ell_0)$ layering strategies denoted by the special levels for $\ell_0 = 12$.

10 Special levels for $\text{SML}_0(12)$	10 Special levels for $\text{SML}_1(12)$
12,7,4,2,1,0	8,4,2,1,0
12,8,4,2,1,0	8,5,2,1,0
12,8,5,2,1,0	8,5,3,1,0
12,8,5,3,1,0	9,5,2,1,0
12,7,3,1,0	9,5,3,1,0
12,7,4,1,0	9,6,3,1,0
12,7,4,2,0	8,4,1,0
12,8,4,1,0	8,4,2,0
12,8,4,2,0	8,5,2,0
12,8,5,2,0	9,5,2,0

1. For a fixed ℓ_0, there may be more than one $\text{SML}_0(\ell_0)$ (resp. $\text{SML}_1(\ell_0)$) strategy which achieves storage of $\#\text{SML}_0(\ell_0)$ (resp. $\#\text{SML}_1(\ell_0)$). Table 2 gives the number of SML strategies for several values of ℓ_0. For $\ell_0 = 12$, Table 3 lists all possible $\text{SML}_0(\ell_0)$ and $\text{SML}_1(\ell_0)$ strategies. There, however, need not be a single layering strategy which minimizes expected header length for all possible values of r. Out of these, one would be interested in the layering that would give the minimum expected header length for most values of r under consideration. The SML strategies reported in Table 1 have this feature.

2. For $\ell_0 = 32$, Tab has been computed and reported in Table 7 in the Appendix. It gives the values of the minimum storage for every $1 \leq \ell_0 \leq 32$ and $1 \leq e \leq \ell_0$. For a particular ℓ_0 and e, it also gives the values of ℓ_1 for which (8) holds. As an example, we see that for $\ell_0 = 32$ and $e = 8$, $\#\text{SML}_0(e, \ell_0) = 172$ and the values of ℓ_1 are 24 and 25. All possible $\text{SML}_0(\ell_0)$ strategies for $1 \leq \ell_0 \leq 32$ can be obtained from this table and the $\text{SML}_1(\ell_0)$ strategies can subsequently be found using (9).

3. As discussed earlier, if the root level is made non-special in an SML_0 strategy, it may not lead to an SML_1 strategy and vice versa. Table 3 shows that while the SML_0 strategy $\boldsymbol{\ell} = (12, 8, 4, 2, 1, 0)$ gives rise to an SML_1 strategy $\boldsymbol{\ell} = (8, 4, 2, 1, 0)$ by making the root level non-special, the SML_0 strategy $\boldsymbol{\ell} = (12, 7, 4, 2, 1, 0)$ does not. On the other hand, the SML_1 strategy $\boldsymbol{\ell} = (9, 5, 2, 1, 0)$ is not generated from an SML_0 strategy.

4. Extensive experiments have shown that for practical values of r, there is no significant difference between the average header lengths of SML_0 and SML_1 strategies that differ at only the root being at a special level or not. For $\ell_0 = 12$ and 16, the reported SML_0 strategy with the root level made non-special turns out to be an SML_1 strategy (as reported in Table 1) with minimum expected header lengths. This supports the theoretical justification described before. However, for $\ell_0 = 20$, it turns out that making the root level of the SML_0 strategy non-special does not give rise to an SML_1 strategy. For $\ell_0 = 24$ and 28, it is again true that making the root level of the reported SML_0 strategy non-special gives rise to an SML_1 strategy. But there are other SML_1 strategies that further reduce the expected header lengths and hence we report those strategies in Table 1.

5. For $\ell_0 = 28$, the HS layering strategy marks the levels 28, 22, 16, 10, 4, 0 as special. Halevy and Shamir [HS02] also consider an alternative layering strategy where the levels 28, 22, 16, 10, 5, 0 are made special. We have denoted this by HS$'$ and reported the average header lengths for this in Table 1.

6. In general, the header length of the HS scheme is smaller than that of SML_0 and SML_1. This is somewhat expected, since user storage in SML is smaller. On the other hand, the user storage is not the only determining factor. The actual layering strategy also plays a role and in some cases the average header length in SML is smaller than that in HS. As a result, in such cases, we see that *both* user storage and average header length are reduced. These are marked in bold and are particularly noticeable for $\ell_0 = 24$ and $\ell_0 = 28$. In the context of AACS standard [AAC], SML_1 for $\ell_0 = 28$ is of particular significance.

3.2 Constrained Minimization of User Storage

From the viewpoint of minimizing communication bandwidth it is of interest to minimize the average header length. This is minimized when the number of keys is maximized which happens for the SD scheme, i.e., when all the levels are considered to be special levels or there is only a single layer. Taking the average header length for the SD scheme as a benchmark, one may ask the question as to how much the user storage can be reduced from that required by the SD scheme without significantly increasing the average header length. The expression for the average header length (as can be derived from (11), (13) and Proposition 2 given later) is rather complicated and it appears quite impossible to have an analytical solution to this question. Instead, we use our average header length computation program to study this behaviour. It turns out that it is indeed possible to significantly reduce the user storage with minimal increase in the average header length.

It has been earlier mentioned that the probability of the root generating an SD subset is small. We build on this intuition. Suppose there are n users and r of them are revoked. In [BSar] it has been shown that the probability that a particular node at level ℓ generates a subset in the header is

$$2(\eta_r(n, 2^{\ell-1}) - \eta_r(n, 2 \times 2^{\ell-1}) - \eta_r(n, 3 \times 2^{\ell-1}) + \eta_r(n, 4 \times 2^{\ell-1}))$$

where $\eta_r(n,x) = (1-x/n)(1-x/(n-1)) \cdots (1-x/(n-r+1))$ if $n > r-1$ else 0. Since there are $2^{\ell_0-\ell}$ nodes at level ℓ, the expected number of subsets arising from all nodes at level ℓ is

$$2^{\ell_0-\ell}(\eta_r(n, 2^{\ell-1}) - \eta_r(n, 2 \times 2^{\ell-1}) - \eta_r(n, 3 \times 2^{\ell-1}) + \eta_r(n, 4 \times 2^{\ell-1})). \tag{10}$$

For a fixed n and r, one can consider the problem of finding ℓ such that this quantity is maximized. One can then make level ℓ special. This will ensure that the subsets generated from this level will not be split due to the layering scheme. Avoiding the splitting of a large number of subsets will have a mitigating effect on the overall header length.

To be able to carry out this strategy, we need to find ℓ for which (10) is maximized. Analytically, this seems to be very difficult to do. Instead we have done extensive experimentation. Empirical values suggest that the maximum occurs for some level $\ell \le \ell_0 - \lfloor \log_2 r \rfloor$. Also, for $\ell > \ell_0 - \lfloor \log_2 r \rfloor$, the value of (10) is quite small.

Based on this empirical evidence we suggest the following layering strategy.

- Make $\ell_0 - \lfloor \log_2 r \rfloor$ special.
- No level $\ell < \ell_0 - \lfloor \log_2 r \rfloor$ is made special. In terms of user storage and expected header length this is equivalent to making all levels $\ell < \ell_0 - \lfloor \log_2 r \rfloor$ to be special.
- The root level is not made special.
- At most one level that is midway between ℓ_0 and $\ell_0 - \lfloor \log_2 r \rfloor$ is made special. While this does not significantly affect header size, it can reduce the storage requirement.

This strategy will ensure that if $\ell \le \ell_0 - \lfloor \log_2 r \rfloor$, then no SD subset generated from level ℓ will be split.

One issue with this strategy is that the value of r will not be known a priori while the layering scheme will have to be decided upon during the design phase itself. A way out is to make an assumption about the minimum number of revoked users that will occur in the steady state operation of the BE scheme. For example, in AACS with 2^{28} users one may assume that in the steady state at least 2^{10} users will be revoked due to data piracy problems.

Suppose that $r_{\min}$ is the minimum number of users that will be revoked during each broadcast. The above layering strategy is used with $r_{\min}$. Suppose now that during a broadcast, the number of users r that is actually revoked is greater than $r_{\min}$. Then from our empirical evidence the level for which the average header length is maximized will be $\ell_0 - \lfloor \log_2 r \rfloor$. Since this value is less than $\ell_0 - \lfloor \log_2 r_{\min} \rfloor$, none of the subsets generated from this level will be split. So, the feature of not splitting a large number of SD subsets is still retained.

Table 4 shows a comparison between the SD scheme, the HS layering scheme and a constrained minimization layering scheme as described above, in terms of both their user storage requirement and

Table 4: Comparison of user storage and average header length for SD, HS-LSD and our constrained minimization layering. The tuples contain header lengths normalized with the SD header lengths corresponding to the values of r in $(r_{\min}, \ldots, r_{\max})$ respectively.

ℓ_0	$r_{\min}$	$r_{\max}$	scheme	special levels	storage	normalized header lengths for $(r_{\min}, \ldots, r_{\max})$
			SD	$12, 0$	78	$(1, \ldots, 1)$
12	2^2	2^8	HS	$12, 8, 4, 0$	42	$(1.69, 1.57, 1.53, 1.45, 1.38, 1.31, 1.26, 1.22, 1.18, 1.15)$
			our	$10, 0$	58	$(1.15, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
			SD	$16, 0$	136	$(1, \ldots, 1)$
16	2^6	2^{12}	HS	$16, 12, 8, 4, 0$	64	$(1.66, 1.57, 1.53, 1.45, 1.38, 1.31, 1.26, 1.22, 1.18, 1.15)$
			our	$10, 0$	76	$(1.14, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
			SD	$20, 0$	210	$(1, \ldots, 1)$
20	2^8	2^{14}	HS	$20, 15, 10, 5, 0$	90	$(1.68, 1.64, 1.64, 1.60, 1.55, 1.50, 1.45, 1.41, 1.38, 1.34)$
			our	$16, 12, 0$	110	$(1.14, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
			SD	$20, 0$	300	$(1, \ldots, 1)$
24	2^{10}	2^{16}	HS	$24, 19, 14, 9, 4, 0$	116	$(1.63, 1.68, 1.63, 1.61, 1.61, 1.61, 1.62, 1.62, 1.63, 1.64)$
			our	$19, 14, 0$	149	$(1.14, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
			SD	$28, 0$	406	$(1, \ldots, 1)$
28	2^{10}	2^{16}	HS	$28, 22, 16, 10, 4, 0$	146	$(1.69, 1.70, 1.75, 1.75, 1.74, 1.73, 1.72, 1.71, 1.70, 1.69)$
			our	$23, 18, 0$	219	$(1.14, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$

the expected header length normalized with respect to the SD scheme. The average header length depends on the number r of revoked users. So, for a given $n = 2^{\ell_0}$, we computed the expected header lengths for 10 equispaced values of r between and including $r_{\min}$ and $r_{\max}$. The values in the table illustrate the point that compared to the SD scheme, the constrained minimization layering schemes substantially reduce the user storage with a small increase in the average header length.

The layering scheme is designed assuming that the number of revoked users is at least $r_{\min}$. What happens if the number of revoked users in an actual broadcast is smaller than $r_{\min}$? Clearly, we cannot expect the average header length to still be almost equal to that of the SD scheme. This effect is shown for some values of r in Table 5. Again the values of the average header length are normalized by that of the corresponding SD scheme. For comparison, we have also provided the average header lengths of the HS layering strategy. It is to be noted that the expected header lengths of the new schemes are mostly better than the HS scheme. As an example, for $n = 2^{24}$, for $r > 6$, our strategy gives smaller expected header lengths than the HS layering strategy. Table 5 shows that for any value of n, the new layering strategies lead to smaller expected header lengths for all $r > 15$.

To summarize, the constrained minimization layering strategy gives low expected header length if $r \geq r_{\min}$. If $r < r_{\min}$, then it is better than HS layering but inferior to the SD scheme. It is to be noted that if r is small, then the absolute size of the header itself is not too large. As a result, the effective transmission overhead of the scheme will never be too high compared to the actual body of the message.

3.3 Tackling Arbitrary Number of Users

In [NNL01] and [HS02], the number of users have been taken to be a power of two, i.e., $n = 2^{\ell_0}$. This is not always convenient as has been argued in details in [BSar]. By modifying the structure of the tree, it is possible to handle arbitrary number of users. This modification is based on the notion of complete binary trees. These are trees where the leaf nodes are at most two different levels and the last level has all its nodes to the left side. An example of a complete subtree accommodating 13 users is shown in Figure 1. In this case $\ell_0 = 4$ and choosing $d = 2$ gives two layers and three special levels as shown in the figure. When the number of users is a power of two, the corresponding tree is called a full binary tree. This difference in terminology between full and complete has been taken from the literature on data structures.

We explain some terminology with respect to Figure 1. The left and the right subtrees of node 3 are the subtrees rooted at nodes 7 and 8 respectively. The sibling subtree of node 3 is the subtree rooted at node 4. The only non-full subtrees are those rooted at nodes 0, 2 and 5. We call the path labelled by the nodes 0, 2, 5 and 11 to be the *dividing path*.

In general given n with $2^{\ell_0 - 1} < n \leq 2^{\ell_0}$, it is possible to accommodate n users as the leaves of

Table 5: Comparison of average header length for $r < r_{\min}$ between HS layering strategy and the constrained minimization layering strategy.

ℓ_0	$r_{\min}$	scheme	special levels	storage	header lengths normalized with the SD scheme
					$r = (1, \mathbf{2}, 3, 4)$
12	2^2	HS	$12, 8, 4, 0$	42	$(1.00, \mathbf{1.74}, 1.72, 1.69)$
		our	$10, 0$	58	$(2.00, \mathbf{1.50}, 1.26, 1.15)$
					$r = (2, 4, 6, 8, 10, \mathbf{12}, 14, 16, 18, 20)$
16	2^6	HS	$12, 8, 4, 0$	64	$(1.75, 1.70, 1.66, 1.63, 1.61, \mathbf{1.60}, 1.60, 1.60, 1.60, 1.61)$
		our	$10, 0$	76	$(1.78, 1.74, 1.70, 1.66, 1.63, \mathbf{1.59}, 1.56, 1.53, 1.50, 1.47)$
					$r = (2, \mathbf{4}, 6, 8, 10, 12, 14, 16, 18, 20)$
20	2^8	HS	$20, 15, 10, 5, 0$	90	$(1.77, \mathbf{1.75}, 1.72, 1.70, 1.68, 1.66, 1.65, 1.64, 1.63, 1.63)$
		our	$16, 12, 0$	110	$(1.77, \mathbf{1.69}, 1.64, 1.61, 1.59, 1.57, 1.56, 1.56, 1.56, 1.56)$
					$r = (2, 4, 6, \mathbf{8}, 10, 12, 14, 16, 18, 20)$
24	2^{10}	HS	$24, 19, 14, 9, 4, 0$	116	$(1.77, 1.75, 1.72, \mathbf{1.70}, 1.68, 1.66, 1.65, 1.64, 1.63, 1.63)$
		our	$19, 14, 0$	149	$(1.79, 1.75, 1.72, \mathbf{1.69}, 1.67, 1.65, 1.64, 1.63, 1.62, 1.61)$
					$r = (2, \mathbf{4}, 6, 8, 10, 12, 14, 16, 18, 20)$
28	2^{10}	HS	$28, 22, 16, 10, 4, 0$	146	$(1.79, \mathbf{1.78}, 1.76, 1.74, 1.73, 1.72, 1.71, 1.70, 1.69, 1.68)$
		our	$23, 18, 0$	219	$(1.79, \mathbf{1.75}, 1.72, 1.69, 1.67, 1.65, 1.64, 1.63, 1.62, 1.61)$

a complete binary tree with n leaves and having the root node at level ℓ_0. The leaves and hence, the users are at either levels 0 or 1. Suppose the layering sequence is $\boldsymbol{\ell} = (\ell_0, \ldots, \ell_e)$ For users at level 0, the storage requirement is $\mathsf{storage}_0(\boldsymbol{\ell})$ while for users at level 1, the storage requirement is $\mathsf{storage}(\boldsymbol{\ell}) - (e + p - 2)$ where p is the number of levels in the bottom-most layer. This reduction is due to the fact that these users need to store one less label for each special level above it and for each level in its last layer. The distribution of labels using the PRG is done as usual.

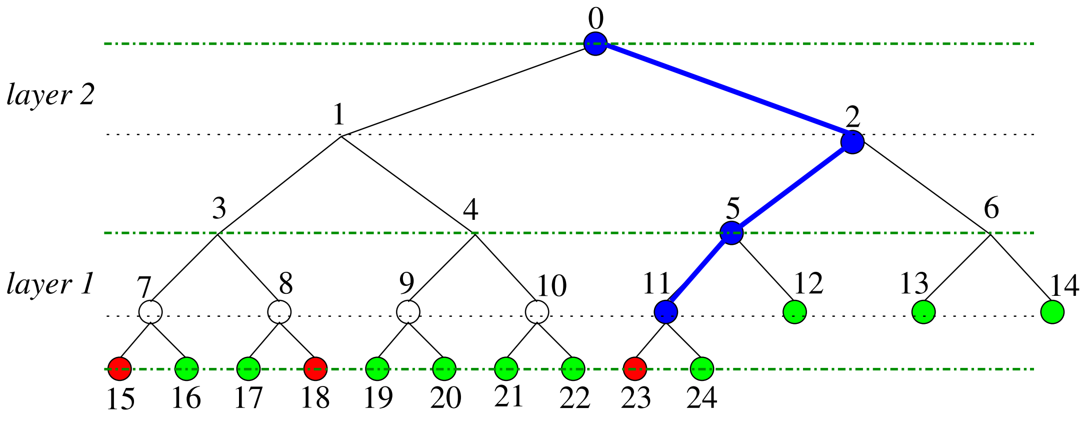

Figure 1: A complete subtree with 13 leaf nodes.

During a broadcast, the actual header generation is done in much the same way. First, as in the SD scheme, the set of non-revoked users is covered exactly by subsets of the form $\mathcal{S}^i \setminus \mathcal{S}^j$ where i is a node in the tree and j is a node in the subtree rooted at i. If i is at a non-special level and j is not in the same layer as i, then this set is further split into $\left(\mathcal{S}^{(i)} \setminus \mathcal{S}^{(k)}\right) \cup \left(\mathcal{S}^{(k)} \setminus \mathcal{S}^{(j)}\right)$ where k is the first node appearing at a special level on the path from i to j.

Complications for complete but non-full trees arise due to the following reason. For some internal nodes, the subtree rooted at it may not be full. All such nodes lie on the dividing path. A node not on the dividing path and at level ℓ is the root of a subtree having either 2^ℓ leaves or $2^{\ell-1}$ leaves accordingly as whether the node is to the left or to the right of the dividing path. As an example, in Figure 1, nodes 3, 4, 5 and 6 are at level 2. Node 5 is on the dividing path and the subtree rooted at node 5 is non-full; nodes 3 and 4 are to the left of 5 and are the roots of subtrees having $2^2 = 4$ leaves; node 6 is to the right of node 5 and the subtree rooted at 6 has 2 leaves.

The LSD scheme is based on full binary trees and the extension to complete binary trees gives rise to the complete tree layered subset difference (CTLSD) scheme. The LSD scheme had improved upon the

SD scheme by reducing the user storage at the cost of almost double the transmission overhead. The CTLSD scheme provides the best trade-offs between the user storage and the transmission overhead for practical values of r. The transmission overhead is almost as good as the (CT)SD scheme, while the user storage is reduced to almost half of it. This is demonstrated at the end of Section 4 using Table 6.

4 Header Length

The main point of the discussion in this section is to obtain an efficient algorithm for computing the expected header length for the CTLSD scheme and hence the LSD scheme. But, before that we state the following bound on the worst case header length.

Proposition 1. *The maximum header length in the CTLSD scheme for n users out of which r are revoked is* $\min\left(4r-2, \left\lceil \frac{n}{2} \right\rceil, n-r\right)$.

Proof. The bound is independent of the actual layering strategy. The upper bound of $2r-1$ for the SD scheme was already given in [NNL01] and in [BSar] it was shown that this also holds for the CTSD scheme. Using the layering strategy, each subset returned by the SD algorithm can split into at most two subsets. So, if the number of SD subsets is at most $2r-1$, then there are at most $4r-2$ subsets.

Suppose the header consists of h subsets out of which h_1 are singleton sets and h_2 sets have 2 or more elements each. For each node in a singleton privileged set, its sibling (if there is one) must be a revoked user. Among all these leaves, there is only one which may not have a sibling that is also a leaf node (and this is the first privileged user from the left at level 1, for odd n). So, for the h_1 privileged users, there are at least h_1-1 other revoked users. This accounts for at least $h_1+h_1-1+2h_2=2h-1$ users. It is now easy to argue that if $h>\lceil n/2\rceil$, then $2h-1$ is greater than n. Since the total number users is n, this cannot happen. So $h\le\lceil n/2\rceil$.

Since each subset in the subset cover will have at least one privileged user, the maximum number of subsets in the header is equal to the number of non-revoked users which is equal to $n-r$. □

The bound of $4r-2$ holds for both the cases when the root is or is not a special level. If the root is a special level the bound of $4r-2$ can be improved to $4r-3$. We first provide a short argument to justify that in the SD scheme if the header length is $2r-1$, then there is a subset of the form $\mathcal{S}^0\setminus\mathcal{S}^j$ in the header. As mentioned earlier, such a subset is added to the header if and only if exactly one of the subtrees of the root node do not contain any revoked user. So, if such a subset is not in the header, then both the subtrees of the root node contain at least one revoked user. Suppose the number of revoked users in these two subtrees are r_1 and r_2 where $r=r_1+r_2$. Applying the bound on the maximum header length, we have the header to be of maximum length $2r_1-1+2r_2-1=2r-2$. So, if the header length is $2r-1$, then there must be a subset of the type $\mathcal{S}^0\setminus\mathcal{S}^j$ in the header. Using the layering strategy, each subset returned by the SD algorithm can split into at most two subsets. So, if the number of SD subsets is at most $2r-2$, then there are at most $4r-4$ subsets. On the other hand, if the number of SD subsets is equal to $2r-1$, then as argued above there must an SD subset of the form $\mathcal{S}^0\setminus\mathcal{S}^j$ in the header. Since the root node 0 is considered to be a special node, this subset will not split while all other subsets may split into two. As a result, there can be at most $4r-3$ subsets in the header.

4.1 Expected Header Length

Assume that the layering strategy is given by $\boldsymbol{\ell}=(\ell_0,\ell_1,\ldots,\ell_e)$. Additionally, the information as to whether the root level is or is not special is also provided as a bit β. If $\beta=0$, then the root node is special and if $\beta=1$, the root node is not special. So, $(\boldsymbol{\ell},\beta)$ provides complete information about the layering strategy. For compactness, we denote this as $\boldsymbol{\ell}_\beta$.

The expected header length is computed under the following random experiment. Out of n users, a set of r users are chosen uniformly at random and these users are revoked. The corresponding header length is then a random variable and let $Y_{n,r}$ denote this header length. We are interested in $E[Y_{n,r}]$. Due to the random revocation of the users, for each internal node i, there arise three possibilities: $\mathcal{S}^{(i)}\setminus\mathcal{S}^{(j)}$ is added to the header; $\left(\mathcal{S}^{(i)}\setminus\mathcal{S}^{(k)}\right)\cup\left(\mathcal{S}^{(k)}\setminus\mathcal{S}^{(j)}\right)$ is added to the header; or nothing is added to the header. So, corresponding to node i, either 0 or 1 or 2 subsets are added to the header. Denote this number by $Y^i_{n,r}$. Then $Y_{n,r} = \sum Y^i_{n,r}$ where the sum is taken over all internal nodes i.

Computing this directly is not convenient. So, we simplify it further. Let $X^i_{n,r}$ be a binary valued random variable which takes the value 1 if and only if there is at least one subset generated from i and let $Z^i_{n,r}$ be another binary valued random variable which takes the value 1 if and only if there are exactly two subsets generated from i. (Note that if i is at a special level, then the probability $Z^i_{n,r} = 1$ is 0.) Then it follows that $Y^i_{n,r} = X^i_{n,r} + Z^i_{n,r}$. The reasoning is as follows. If i generates no subset, then both sides are zero; if exactly one subset is generated, then $Y^i_{n,r}$ and $X^i_{n,r}$ are both 1 but, $Z^i_{n,r}$ is 0; if exactly two subsets are generated then $Y^i_{n,r}$ is 2 and both $X^i_{n,r}$ and $Z^i_{n,r}$ are 1. By linearity of expectation, we have

$$E[Y_{n,r}] \;=\; E\left[\sum Y^i_{n,r}\right] = \sum E\left[X^i_{n,r} + Z^i_{n,r}\right] = \sum E\left[X^i_{n,r}\right] + \sum E\left[Z^i_{n,r}\right]. \tag{11}$$

The sum is over all internal nodes i of the tree. The quantity $\sum X^i_{n,r}$ is exactly the expected header length obtained using the SD algorithm. This is because i generates at least one subset if and only if the SD algorithm results in i generating a subset. Let $X_{n,r} = \sum X^i_{n,r}$ and $Z_{n,r} = \sum Z^i_{n,r}$. So,

$$E\left[Y_{n,r}\right] = E\left[X_{n,r}\right] + E\left[Z_{n,r}\right]. \tag{12}$$

An algorithm for computing $E[X_{n,r}]$ has been already developed in [BSar]. So, it only remains to determine $E[Z_{n,r}]$.

Given n and a layering sequence $\boldsymbol{\ell}_\beta$ we define the set $\mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$ to consist of pairs of nodes (i, j) such that i is not at a special level and j is in the subtree rooted at i but not in the same layer as i. So, whenever an SD subset $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$ is such that $(i, j) \in \mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$, it is split into two subsets. If i is at level ℓ, then there are at most $\ell - 1$ values of level for j such that (i, j) is in $\mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$.

Let i be at a non-special level and let j be not in the same layer as i. Define the binary valued random variable $W^{i,j}_{n,r}$ to take the value 1 if and only if the SD algorithm returns the subset $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$ to the header, in which case the LSD algorithm will split this subset into two sets. So, we have $Z^i_{n,r} = \sum_{(i,j) \in \mathsf{SubsetsForSplit}(n,\boldsymbol{\ell}_\beta)} W^{i,j}_{n,r}$. Again by linearity of expectation, the task reduces to computing $E[W^{i,j}_{n,r}]$. Since this is a binary valued random variable, $E[W^{i,j}_{n,r}] = \Pr[W^{i,j}_{n,r} = 1]$. So,

$$E[Z_{n,r}] \;=\; \sum_i E[Z^i_{n,r}] = \sum_i \sum_{(i,j) \in \mathsf{SubsetsForSplit}(n,\boldsymbol{\ell}_\beta)} \Pr[W^{i,j}_{n,r} = 1]. \tag{13}$$

Here the first sum is over all nodes i at non-special levels. For a fixed i and j, we show how to compute $\Pr[W^{i,j}_{n,r} = 1]$. To do this, we need to characterize the event $W^{i,j}_{n,r} = 1$ for a pair $(i, j) \in \mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$. This event occurs if and only if the following conditions hold.

- Node i is either the root (in which case it does not have any sibling tree) or the sibling tree of i has at least one revoked user among its leaves.
- Either j is a leaf and is revoked or both subtrees of j have at least one revoked user among its leaves.
- There are no revoked users in the set $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$.

Define the following events:

R^j_{lt}	: there is at least one revoked user in the left subtree of j;
R^j_{rt}	: there is at least one revoked user in the right subtree of j;
R^i_{sb}	: there is at least one revoked user in the sibling subtree of i;
$R^{i,j}_{rm}$	: there is at least one revoked user in the set $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$.

Let $(i, j) \in \mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$. Suppose i is not the root. If j is not a leaf node, the event $W^{i,j}_{n,r} = 1$ is equivalent to the event $R^i_{sb} \wedge \overline{R^{i,j}_{rm}} \wedge R^j_{lt} \wedge R^j_{rt}$. If j is a leaf node, the event $W^{i,j}_{n,r} = 1$ is equivalent to the event $R^{i}_{sb} \wedge \overline{R^{i,j}_{rm}}$. Now suppose i is the root and is not special (i.e., $\beta = 1$). If j is not a leaf, then the event $W^{i,j}_{n,r} = 1$ is equivalent to $\overline{R^{i,j}_{rm}} \wedge R^{j}_{lt} \wedge R^{j}_{rt}$. If j is a leaf, then this can happen only if there is a single revoked user. So, for $r = 1$, the probability of $W^{i,j}_{n,r} = 1$ is 1 and for $r \geq 2$, the probability of $W^{i,j}_{n,r} = 1$ is 0.

Let λ_i (resp. λ_j; λ_s) be the number of leaves in the subtree rooted at i (resp. j; the sibling subtree of i). Similarly, let λ_{2j+1} and λ_{2j+2} respectively be the number of leaves in the left and right subtrees of j. So, $\lambda_j = \lambda_{2j+1} + \lambda_{2j+2}$. The number of leaves in the set $\mathcal{S}^{(i)} \setminus \mathcal{S}^{(j)}$ is $\lambda_i - \lambda_j$. Note that since we are dealing with arbitrary number of users, the subtrees that are being considered are not necessarily full. So, the values of the λ's are not necessarily powers of two.

Fix t users and consider the probability $\eta_r(n,t)$ that in the random experiment none of these t users have been chosen. Recall that the random experiment is to choose r users uniformly and without replacement from the set of n users. As discussed earlier

$$\eta_r(n,t) = \left(1 - \frac{t}{n}\right) \times \left(1 - \frac{t}{n-1}\right) \times \cdots \times \left(1 - \frac{t}{n-r+1}\right).$$

This makes it convenient to express the probability that none among a set of users of certain size is revoked. For example, the probability of $\overline{R^{j}_{lt}}$ is $\eta_r(n, \lambda_{2j+1})$. Similarly, the probability of the event $\overline{R^{j}_{lt}} \wedge \overline{R^{i,j}_{rm}}$ is $\eta_r(n, \lambda_{2j+1} + \lambda_i - \lambda_j) = \eta_r(n, \lambda_i - \lambda_{2j+2})$. Such calculations will be used in what follows.

Proposition 2. *Let i and j be nodes such that $(i,j) \in \mathsf{SubsetsForSplit}(n, \boldsymbol{\ell}_\beta)$.*

- *If i is the root and j is a leaf, then $\Pr[W^{i,j}_{n,r} = 1] = 1$ if $r = 1$ and $\Pr[W^{i,j}_{n,r} = 1] = 0$ if $r \geq 2$.*
- *If i is the root and j is not a leaf, then*

$$\begin{aligned}\Pr[W^{i,j}_{n,r} = 1] = {} & \eta_r(n, \lambda_i - \lambda_j) - \eta_r(n, \lambda_{2j+1} + \lambda_i - \lambda_j) - \eta_r(n, \lambda_{2j+2} + \lambda_i - \lambda_j) \\ & + \eta_r(n, \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j). \end{aligned} \tag{14}$$

- *If i is not the root and j is a leaf, then*

$$\Pr[W^{i,j}_{n,r} = 1] = \eta_r(n, \lambda_i - \lambda_j) - \eta_r(n, \lambda_s + \lambda_i - \lambda_j). \tag{15}$$

- *If i is not the root and j is not a leaf, then*

$$\begin{aligned}\Pr[W^{i,j}_{n,r} = 1] = {} & \eta_r(n, \lambda_i - \lambda_j) - \eta_r(n, \lambda_s + \lambda_i - \lambda_j) - \eta_r(n, \lambda_{2j+1} + \lambda_i - \lambda_j) \\ & - \eta_r(n, \lambda_{2j+2} + \lambda_i - \lambda_j) \\ & + \eta_r(n, \lambda_s + \lambda_{2j+1} + \lambda_i - \lambda_j) + \eta_r(n, \lambda_s + \lambda_{2j+2} + \lambda_i - \lambda_j) \\ & + \eta_r(n, \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j) - \eta_r(n, \lambda_s + \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j). \end{aligned} \tag{16}$$

Proof. We consider the case when i is not the root and j is not a leaf. The other cases are similar. When i is not the root and j is not a leaf, the event $W^{i,j}_{n,r} = 1$ is equivalent to the event $R^{i,j}_{sb} \wedge \overline{R^{i,j}_{rm}} \wedge R^{i,j}_{lt} \wedge R^{i,j}_{rt}$.

We now compute as follows.

$$
\begin{aligned}
\Pr[R^{i,j}_{sb} \wedge \overline{R^{i,j}_{rm}} \wedge R^{i,j}_{lt} \wedge R^{i,j}_{rt}] &= \Pr[R^{i,j}_{sb} \wedge R^{i,j}_{lt} \wedge R^{i,j}_{rt} | \overline{R^{i,j}_{rm}}] \times \Pr[\overline{R^{i,j}_{rm}}] \\
&= \Big(1 - \Pr[\overline{R^{i,j}_{sb} \wedge R^{i,j}_{lt} \wedge R^{i,j}_{rt}} | \overline{R^{i,j}_{rm}}]\Big) \times \Pr[\overline{R^{i,j}_{rm}}] \\
&= (1 - \Pr[\overline{R^{i,j}_{sb}} | \overline{R^{i,j}_{rm}}] - \Pr[\overline{R^{i,j}_{lt}} | \overline{R^{i,j}_{rm}}] - \Pr[\overline{R^{i,j}_{rt}} | \overline{R^{i,j}_{rm}}] \\
&\quad + \Pr[\overline{R^{i,j}_{sb}} \wedge \overline{R^{i,j}_{lt}} | \overline{R^{i,j}_{rm}}] + \Pr[\overline{R^{i,j}_{sb}} \wedge \overline{R^{i,j}_{rt}} | \overline{R^{i,j}_{rm}}] + \Pr[\overline{R^{i,j}_{lt}} \wedge \overline{R^{i,j}_{rt}} | \overline{R^{i,j}_{rm}}] \\
&\quad - \Pr[\overline{R^{i,j}_{sb}} \wedge \overline{R^{i,j}_{lt}} \wedge \overline{R^{i,j}_{rt}} | \overline{R^{i,j}_{rm}}]) \times \Pr[\overline{R^{i,j}_{rm}}] \\
&= (\Pr[\overline{R^{i,j}_{rm}}] - \Pr[\overline{R^{i,j}_{sb}} \wedge \overline{R^{i,j}_{rm}}] - \Pr[\overline{R^{i,j}_{lt}} \wedge \overline{R^{i,j}_{rm}}] - \Pr[\overline{R^{i,j}_{rt}} \wedge \overline{R^{i,j}_{rm}}] \\
&\quad + \Pr[\overline{R^{i,j}_{sb}} \wedge \overline{R^{i,j}_{lt}} \wedge \overline{R^{i,j}_{rm}}] + \Pr[\overline{R^{i,j}_{sb}} \wedge \overline{R^{i,j}_{rt}} \wedge \overline{R^{i,j}_{rm}}] + \Pr[\overline{R^{i,j}_{lt}} \wedge \overline{R^{i,j}_{rt}} \wedge \overline{R^{i,j}_{rm}}] \\
&\quad - \Pr[\overline{R^{i,j}_{sb}} \wedge \overline{R^{i,j}_{lt}} \wedge \overline{R^{i,j}_{rt}} \wedge \overline{R^{i,j}_{rm}}]) \\
&= \eta_r(n, \lambda_i - \lambda_j) - \eta_r(n, \lambda_s + \lambda_i - \lambda_j) - \eta_r(n, \lambda_{2j+1} + \lambda_i - \lambda_j) \\
&\quad - \eta_r(n, \lambda_{2j+2} + \lambda_i - \lambda_j) \\
&\quad + \eta_r(n, \lambda_s + \lambda_{2j+1} + \lambda_i - \lambda_j) + \eta_r(n, \lambda_s + \lambda_{2j+2} + \lambda_i - \lambda_j) \\
&\quad + \eta_r(n, \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j) - \eta_r(n, \lambda_s + \lambda_{2j+1} + \lambda_{2j+2} + \lambda_i - \lambda_j).
\end{aligned}
$$

The above expression is obtained by conditioning on the event $\overline{R^{i,j}_{rm}}$ and so for the computation to go through one needs to assume that the probability of this event is positive. In the case where this probability is zero, one can directly verify that the probabilities on both sides are zero. □

Algorithm to compute $Z_{n,r}$: For any fixed $(i,j) \in \mathsf{SubsetsForSplits}(n, \boldsymbol{\ell}_\beta)$, Theorem 2 provides a method for computing $\Pr[W^{i,j}_{n,r} = 1]$. Each of the η expressions can be computed using r multiplications and since there are a constant number of η's, the value of $\Pr[W^{i,j}_{n,r} = 1]$ can be computed using $O(r)$ multiplications. Using (13) this immediately gives a method for computing $Z_{n,r}$. Doing this directly, however, is not very efficient. The first sum in (13) is over all possible nodes i and the second sum is over the relevant j which are paired with i. Since the number of nodes is $O(n)$, a direct computation will lead to an algorithm whose running time is $O(rn^2)$.

This can be significantly improved. To explain the idea, first consider n to be a power of two so that the tree is a full binary tree. Fix a non-special node i and consider all possible j for which the second sum in (13) has to be evaluated. From the expression for $\Pr[W^{i,j}_{n,r} = 1]$ it is easy to note that for a fixed (n and r and) i, the value of $\Pr[W^{i,j}_{n,r} = 1]$ is determined only by the number of leaves in the subtree rooted at j and consequently the number of leaves in the left and the right subtrees of j. Since the tree is full, these values depend only on the value of the level of node j. So, for each appropriate level below i, one can compute the value of $\Pr[W^{i,j}_{n,r} = 1]$ for one particular j at that level and then multiply by the number of nodes in the subtree rooted at i at the level of j. As a result, the second sum in (13) can be computed in $O(r \log \lambda_i)$ time where λ_i is the number of leaves in the subtree rooted at i so that $\log \lambda_i$ is the level number of i. Since $\lambda_i \leq n$, the second sum in (13) can be computed using $O(r \log n)$ time.

Consider now the first sum in (13) (and still assume that n is a power of two). Again, it is easy to note that the value of $E[Z^i_{n,r}]$ is determined by the value of the level number of i. So, for each appropriate level, one can compute $E[Z^i_{n,r}]$ for one i and then multiply by the number of nodes at that level. As a result, computing $E[Z_{n,r}]$ requires a total of $O(r \log^2 n)$ multiplications.

If n is not a power of two, then the tree is a complete but, non-full tree and we need to revise the above description. The idea that all nodes at the same level contribute the same value does not hold any more. This is because the number of leaves in the subtrees rooted at nodes at the same level can be different. There is however, a way out which is based on the idea of the dividing path. One may recollect that the dividing path joins all nodes that are roots of non-full subtrees. All nodes at the same level and on the same side of the dividing path have the same number of leaf nodes. So, for each level, we compute separately for three cases: for nodes to the left of the dividing path; for the node on the dividing path; and for nodes to the right of the dividing path. For nodes at the same level and on the same side of the dividing path, we compute $\Pr[W^{i,j}_{n,r} = 1]$ once and multiply by the number of nodes

n	scheme	special layers	storage	$r_{\min}$	$r_{\max}$	header length normalized by CTSD
10^3	CTSD	10,0	55	2^2	2^8	$(1, \ldots, 1)$
	CTLSD	8,0	**39**	2^2	2^8	$(1.09, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^4	CTSD	10,0	105	2^4	2^{10}	$(1, \ldots, 1)$
	CTLSD	8,0	**65**	2^4	2^{10}	$(1.04, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^5	CTSD	17,0	153	2^6	2^{12}	$(1, \ldots, 1)$
	CTLSD	11,0	**87**	2^6	2^{12}	$(1.08, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^6	CTSD	20,0	210	2^8	2^{14}	$(1, \ldots, 1)$
	CTLSD	16,12,0	**110**	2^8	2^{14}	$(1.13, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^7	CTSD	24,0	300	2^{10}	2^{16}	$(1, \ldots, 1)$
	CTLSD	19,14,0	**149**	2^{10}	2^{16}	$(1.04, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^8	CTSD	27,0	378	2^{10}	2^{16}	$(1, \ldots, 1)$
	CTLSD	22,17,0	**200**	2^{10}	2^{16}	$(1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$
10^9	CTSD	30,0	465	2^{10}	2^{16}	$(1, \ldots, 1)$
	CTLSD	25,20,0	**260**	2^{10}	2^{16}	$(1.12, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00)$

Table 6: Comparison of the storage and the expected header lengths for the CTSD and the CTLSD schemes.

satisfying this condition. Similarly the computation of $E[Z^i_{n,r}]$ is carried out. Overall, the complexity of the algorithm is still $O(r \log^2 n)$.

There is one complication that we have not explained. This is the problem of characterizing the dividing path and counting the number of nodes at the same level and on the same side of the dividing path. It turns out that given the value of n, this can always be done. The details are provided in [BSar] and so are omitted here. We have incorporated these in our implementation of the algorithm to compute expected header length given any value of n and r.

The expected header length of the CTLSD method is $E[Y_{n,r}]$. As given in (12), this quantity is equal to the sum of $E[X_{n,r}]$ and $E[Z_{n,r}]$. We have shown that $E[Z_{n,r}]$ can be computed in $O(r \log^2 n)$ time. The quantity $E[X_{n,r}]$ is the expected header length of the CTSD scheme and can be computed in $O(r \log n)$ time [BSar]. So, the overall complexity of the algorithm is $O(r \log^2 n)$.

Table 6 provides some examples of running the algorithm for computing expected header length for non-full trees using the CTSD and the CTLSD schemes. The chosen values of r are 10 equispaced values between $r_{\min}$ and $r_{\max}$ for the respective n. The CTLSD method is run by adopting the constrained minimization layering strategy where all levels including and below $\ell_0 - \lfloor \log_2 r_{\min} \rfloor$ are considered to be in one layer. The expected header length of the CTLSD method is almost similar to the CTSD scheme while the user storage requirement is a little more than half of the CTSD scheme. Hence, with an assumption on the minimum number of users, the CTLSD scheme with the constrained minimization layering strategy would be the more practical choice.

Since the CTLSD scheme subsumes the LSD scheme, this algorithm computes the expected header length for the LSD scheme too. In [HS02], it was mentioned that the expected header length for their layering scheme, i.e; HS layering is around $2r$. As we have seen earlier, by suitably placing the special levels, this can be brought down significantly to about the expected header length of the SD scheme. On the other hand, for the HS layering, the expected header length can also be somewhat larger than $2r$. For example, for $l_0 = 28$ and $r = 2$, the expected header length is $2.23r$.

5 Conclusion

In this work, we have suggested new layering strategies for the SD scheme. At one end we have shown that it is possible to decrease the user storage below that obtained by Halevy and Shamir [HS02]. At the other end, we have shown that it is possible to attain header length very close to that of the SD scheme while still requiring a significantly smaller number of keys. The LSD scheme is extended to handle arbitrary number of users leading to the CTLSD scheme. We have obtained an efficient algorithm to compute the expected header length in the CTLSD scheme. Our analysis of different scenarios is made possible by using this algorithm.

References

[AAC] AACS. Advanced Access Content System, `http://www.aacsla.com`.

[AK08] Per Austrin and Gunnar Kreitz. Lower bounds for subset cover based broadcast encryption. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2008.

[AKI03] Nuttapong Attrapadung, Kazukuni Kobara, and Hideki Imai. Sequential key derivation patterns for broadcast encryption and key predistribution schemes. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 374–391. Springer, 2003.

[Ber91] Shimshon Berkovits. How to broadcast a secret. In Donald W. Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 535–541. Springer, 1991.

[BF99] Dan Boneh and Matthew K. Franklin. An efficient public key traitor tracing scheme. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 1999.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2005.

[BSar] Sanjay Bhattacherjee and Palash Sarkar. Complete tree subset difference broadcast encryption scheme and its analysis. *Designs, Codes and Cryptography*, to appear.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1994.

[DF03] Yevgeniy Dodis and Nelly Fazio. Public key trace and revoke scheme secure against adaptive chosen ciphertext attack. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 100–115. Springer, 2003.

[EOPR08] Christopher Eagle, Mohamed Omar, Daniel Panario, and Bruce Richmond. Distribution of the number of encryptions in revocation schemes for stateless receivers. In Uwe Roesler, Jan Spitzmann, and Marie-Christine Ceulemans, editors, *Fifth Colloquium on Mathematics and Computer Science*, volume AI of *DMTCS Proceedings*, pages 195–206. Discrete Mathematics and Theoretical Computer Science, 2008.

[FN93] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer, 1993.

[FT01] Amos Fiat and Tamir Tassa. Dynamic traitor tracing. *J. Cryptology*, 14(3):211–223, 2001.

[GST04] Michael T. Goodrich, Jonathan Z. Sun, and Roberto Tamassia. Efficient tree-based revocation in groups of low-state devices. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 511–527. Springer, 2004.

[HS02] Dani Halevy and Adi Shamir. The LSD broadcast encryption scheme. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 47–60. Springer, 2002.

[JHC$^+$05] Nam-Su Jho, Jung Yeon Hwang, Jung Hee Cheon, Myung-Hwan Kim, Dong Hoon Lee, and Eun Sun Yoo. One-way chain based broadcast encryption schemes. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 2005.

[KY01] Aggelos Kiayias and Moti Yung. Self protecting pirates and black-box traitor tracing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 63–79. Springer, 2001.

[LS98] Michael Luby and Jessica Staddon. Combinatorial bounds for broadcast encryption. In Kaisa Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 512–526. Springer, 1998.

[MMW09] Thomas Martin, Keith M. Martin, and Peter R. Wild. Establishing the broadcast efficiency of the subset difference revocation scheme. *Des. Codes Cryptography*, 51(3):315–334, 2009.

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001.

[NP98] Moni Naor and Benny Pinkas. Threshold traitor tracing. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 502–517. Springer, 1998.

[PB06] E. C. Park and Ian F. Blake. On the mean number of encryptions for tree-based broadcast encryption schemes. *J. Discrete Algorithms*, 4(2):215–238, 2006.

[PGM04] Carles Padró, Ignacio Gracia, and Sebastià Martín Molleví. Improving the trade-off between storage and communication in broadcast encryption schemes. *Discrete Applied Mathematics*, 143(1-3):213–220, 2004.

[PGMM02] Carles Padró, Ignacio Gracia, Sebastià Martín Molleví, and Paz Morillo. Linear key predistribution schemes. *Des. Codes Cryptography*, 25(3):281–298, 2002.

[PGMM03] Carles Padró, Ignacio Gracia, Sebastià Martín Molleví, and Paz Morillo. Linear broadcast encryption schemes. *Discrete Applied Mathematics*, 128(1):223–238, 2003.

[SSW01] Alice Silverberg, Jessica Staddon, and Judy L. Walker. Efficient traitor tracing algorithms using list decoding. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 175–192. Springer, 2001.

[Sti97] Douglas R. Stinson. On some methods for unconditionally secure key distribution and broadcast encryption. *Des. Codes Cryptography*, 12(3):215–243, 1997.

[SW98] Douglas R. Stinson and Ruizhong Wei. Combinatorial properties and constructions of traceability schemes and frameproof codes. *SIAM J. Discrete Math.*, 11(1):41–53, 1998.

Appendices

A Storage Minimal Layering: Practical Examples

Table 7 provides values of $\#\mathrm{SML}_0(e, \ell_0)$ for all practical values of ℓ_0 and e.

Table 7: $\#\mathrm{SML}_0(e,\ell_0)$ and $\Lambda(e,\ell_0)$ for $1 \le \ell_0 \le 32$ and $1 \le e \le \ell_0$.

ℓ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1(0)	3(0)	6(0)	10(0)	15(0)	21(0)	28(0)	36(0)	45(0)	55(0)	66(0)	78(0)	91(0)	105(0)	120(0)	136(0)
2		3(1)	5(1)	8(1,2)	11(2)	15(2,3)	19(3)	24(3,4)	29(4)	35(4,5)	41(5)	48(5,6)	55(6)	63(6,7)	71(7)	80(7,8)
3			6(2)	8(2)	11(2,3)	14(3)	18(3,4)	22(4,5)	26(5)	31(5,6)	36(6,7)	41(7)	47(7,8)	53(8,9)	59(9)	66(9,10)
4				10(3)	12(3)	15(3,4)	18(4)	22(4,5)	26(5,6)	30(6)	35(6,7)	40(7,8)	45(8,9)	50(9)	56(9,10)	62(10,11)
5					15(4)	17(4)	20(4,5)	23(5)	27(5,6)	31(6,7)	35(7)	40(7,8)	45(8,9)	50(9,10)	55(10)	61(10,11)
6						21(5)	23(5)	26(5,6)	29(6)	33(6,7)	37(7,8)	41(8)	46(8,9)	51(9,10)	56(10,11)	61(11)
7							28(6)	30(6)	33(6,7)	36(7)	40(7,8)	44(8,9)	48(9)	53(9,10)	58(10,11)	63(11,12)
8								36(7)	38(7)	41(7,8)	44(8)	48(8,9)	52(9,10)	56(10)	61(10,11)	66(11,12)
9									45(8)	47(8)	50(8,9)	53(9)	57(9,10)	61(10,11)	65(11)	70(11,12)
10										55(9)	57(9)	60(9,10)	63(10)	67(10,11)	71(11,12)	75(12)
11											66(10)	68(10)	71(10,11)	74(11)	78(11,12)	82(12,13)
12												78(11)	80(11)	83(11,12)	86(12)	90(12,13)
13													91(12)	93(12)	96(12,13)	99(13)
14														105(13)	107(13)	110(13,14)
15															120(14)	122(14)
16																136(15)

ℓ	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	153(0)	171(0)	190(0)	210(0)	231(0)	253(0)	276(0)	300(0)	325(0)	351(0)	378(0)	406(0)	435(0)	465(0)	496(0)	528(0)
2	89(8)	99(8,9)	109(9)	120(9,10)	131(10)	143(10,11)	155(11)	168(11,12)	181(12)	195(12,13)	209(13)	224(13,14)	239(14)	255(14,15)	271(15)	288(15,16)
3	73(10,11)	80(11)	88(11,12)	96(12,13)	104(13)	113(13,14)	122(14,15)	131(15)	141(15,16)	151(16,17)	161(17)	172(17,18)	183(18,19)	194(19)	206(19,20)	218(20,21)
4	68(11,12)	74(12)	81(12,13)	88(13,14)	95(14,15)	102(15)	110(15,16)	118(16,17)	126(17,18)	134(18)	143(18,19)	152(19,20)	161(20,21)	170(21)	180(21,22)	190(22,23)
5	67(11,12)	73(12,13)	79(13,14)	85(14)	92(14,15)	99(15,16)	106(16,17)	113(17,18)	120(18)	128(18,19)	136(19,20)	144(20,21)	152(21,22)	160(22)	169(22,23)	178(23,24)
6	67(11,12)	73(12,13)	79(13,14)	85(14,15)	91(15)	98(15,16)	105(16,17)	112(17,18)	119(18,19)	126(19,20)	133(20)	141(20,21)	149(21,22)	157(22,23)	165(23,24)	173(24,25)
7	68(12)	74(12,13)	80(13,14)	86(14,15)	92(15,16)	98(16)	105(16,17)	112(17,18)	119(18,19)	126(19,20)	133(20,21)	140(21)	148(21,22)	156(22,23)	164(23,24)	172(24,25)
8	71(12,13)	76(13)	82(13,14)	88(14,15)	94(15,16)	100(16,17)	106(17)	113(17,18)	120(18,19)	127(19,20)	134(20,21)	141(21,22)	148(22)	156(22,23)	164(23,24)	172(24,25)
9	75(12,13)	80(13,14)	85(14)	91(14,15)	97(15,16)	103(16,17)	109(17,18)	115(18)	122(18,19)	129(19,20)	136(20,21)	143(21,22)	150(22,23)	157(23)	165(23,24)	173(24,25)
10	80(12,13)	85(13,14)	90(14,15)	95(15)	101(15,16)	107(16,17)	113(17,18)	119(18,19)	125(19)	132(19,20)	139(20,21)	146(21,22)	153(22,23)	160(23,24)	167(24)	175(24,25)
11	86(13)	91(13,14)	96(14,15)	101(15,16)	106(16)	112(16,17)	118(17,18)	124(18,19)	130(19,20)	136(20)	143(20,21)	150(21,22)	157(22,23)	164(23,24)	171(24,25)	178(25)
12	94(13,14)	98(14)	103(14,15)	108(15,16)	113(16,17)	118(17)	124(17,18)	130(18,19)	136(19,20)	142(20,21)	148(21)	155(21,22)	162(22,23)	169(23,24)	176(24,25)	183(25,26)
13	103(13,14)	107(14,15)	111(15)	116(15,16)	121(16,17)	126(17,18)	131(18)	137(18,19)	143(19,20)	149(20,21)	155(21,22)	161(22)	168(22,23)	175(23,24)	182(24,25)	189(25,26)
14	113(14)	117(14,15)	121(15,16)	125(16)	130(16,17)	135(17,18)	140(18,19)	145(19)	151(19,20)	157(20,21)	163(21,22)	169(22,23)	175(23)	182(23,24)	189(24,25)	196(25,26)
15	125(14,15)	128(15)	132(15,16)	136(16,17)	140(17)	145(17,18)	150(18,19)	155(19,20)	160(20)	166(20,21)	172(21,22)	178(22,23)	184(23,24)	190(24)	197(24,25)	204(25,26)
16	138(15)	141(15,16)	144(16)	148(16,17)	152(17,18)	156(18)	161(18,19)	166(19,20)	171(20,21)	176(21)	182(21,22)	188(22,23)	194(23,24)	200(24,25)	206(25)	213(25,26)
17	153(16)	155(16)	158(16,17)	161(17)	165(17,18)	169(18,19)	173(19)	178(19,20)	183(20,21)	188(21,22)	193(22)	199(22,23)	205(23,24)	211(24,25)	217(25,26)	223(26)
18		171(17)	173(17)	176(17,18)	179(18)	183(18,19)	187(19,20)	191(20)	196(20,21)	201(21,22)	206(22,23)	211(23)	217(23,24)	223(24,25)	229(25,26)	235(26,27)
19			190(18)	192(18)	195(18,19)	198(19)	202(19,20)	206(20,21)	210(21)	215(21,22)	220(22,23)	225(23,24)	230(24)	236(24,25)	242(25,26)	248(26,27)
20				210(19)	212(19)	215(19,20)	218(20)	222(20,21)	226(21,22)	230(22)	235(22,23)	240(23,24)	245(24,25)	250(25)	256(25,26)	262(26,27)
21					231(20)	233(20)	236(20,21)	239(21)	243(21,22)	247(22,23)	251(23)	256(23,24)	261(24,25)	266(25,26)	271(26)	277(26,27)
22						253(21)	255(21)	258(21,22)	261(22)	265(22,23)	269(23,24)	273(24)	278(24,25)	283(25,26)	288(26,27)	293(27)
23							276(22)	278(22)	281(22,23)	284(23)	288(23,24)	292(24,25)	296(25)	301(25,26)	306(26,27)	311(27,28)
24								300(23)	302(23)	305(23,24)	308(24)	312(24,25)	316(25,26)	320(26)	325(26,27)	330(27,28)
25									325(24)	327(24)	330(24,25)	333(25)	337(25,26)	341(26,27)	345(27)	350(27,28)
26										351(25)	353(25)	356(25,26)	359(26)	363(26,27)	367(27,28)	371(28)
27											378(26)	380(26)	383(26,27)	386(27)	390(27,28)	394(28,29)
28												406(27)	408(27)	411(27,28)	414(28)	418(28,29)
29													435(28)	437(28)	440(28,29)	443(29)
30														465(29)	467(29)	470(29,30)
31															496(30)	498(30)
32																528(31)