Complete Tree Subset Difference Broadcast Encryption Scheme and its Analysis

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

The Subset Difference (SD) method proposed by Naor, Naor and Lotspiech is the most popular broadcast encryption (BE) scheme. It is suitable for real-time applications like Pay-TV and has been suggested for use by the AACS standard for digital rights management in Blu-Ray and HD-DVD discs. The SD method assumes the number of users to be a power of two. We propose the Complete Tree Subset Difference (CTSD) method that allows the system to support an arbitrary number of users. In particular, it subsumes the SD method and all results proved for the CTSD method also hold for the SD method. Recurrences are obtained for the CTSD scheme to count the number, $N(n, r, h)$, of possible ways r users in the system of n users can be revoked to result in a transmission overhead or header length of h. The recurrences lead to a polynomial time dynamic programming algorithm for computing $N(n, r, h)$. Further, they provide bounds on the maximum possible header length. A probabilistic analysis is performed to obtain an $O(r \log n)$ time algorithm to compute the expected header length in the CTSD scheme. Further, for the SD scheme we obtain an explicit limiting upper bound on the expected header length.

Keywords: Broadcast encryption; subset difference; combinatorial analysis; recurrence; probabilistic analysis; expected header length; transmission overhead; asymptotic analysis

1 Introduction

A Broadcast Encryption (BE) scheme allows a centre to efficiently broadcast encrypted information so that only an intended set of users can recover the message. Before the system starts to work, the users are given some secret information. This could be the actual decryption keys or some information from which it can derive the decryption keys. A user uses this information for decrypting relevant encrypted digital content. Copyright protection using Digital Rights Management (DRM) techniques is an important application of BE. The application of BE systems is pretty wide in the implementation of DRM for content protection in digital data distribution technologies such as pay-TV, Internet or mobile video broadcast, optical discs, etc.

In a typical BE scheme, the entire digital data to be broadcast is divided into blocks. Each such block is called a message. Each message to be broadcast, is encrypted using a unique key called a *session key*. The session key in turn, is encrypted a number of times using user keys and these multiple encryptions of the session key are sent as the *header* of the encrypted message. The transmission overhead of the scheme is determined by the number of encryptions of the session key in the header. This is called the *header length* and we denote this quantity by h.

In a fully resilient scheme, even if an adversary has the decryption keys of all the remaining non-privileged users in the system, it will not be able to correctly decrypt the content. The non-privileged users are called *revoked* users. A crucial requirement for a BE scheme is that it should facilitate dynamic revocation of decryption privilege from any subset of users at any point of time. The decision could be based on their subscription or privilege status.

In real-time scenarios like Pay-TV, Internet or mobile video broadcast, the number of users can vary from a few thousands to millions. For other real-time applications of BE like broadcasting secret instructions to military outposts from a base station, the number of users will be a few hundreds. The BE scheme that is used in real time scenarios as above, has to be efficient in terms of the transmission overhead associated with each message as also the encryption and decryption times and storage of user keys. For non-real-time applications like content protection in Blu-Ray discs and HD-DVDs, the requirements from a BE scheme are somewhat different. Here, the transmission overhead is the additional information stored in the physical media that is used for decrypting the content. Storage space in discs is no more a constraint nowadays. Further, since encryption does not happen in real-time, improving the encryption time is also not very important. On the other hand, reducing the user storage and decryption time is still important.

Broadcast Encryption was introduced in [Ber91] followed by [FN93]. There have been several works in this area [Sti97, SW98] since then, but the most popular scheme out of these is the tree-based Subset Difference (SD) method of [NNL01]. Since it is a symmetric key based scheme, it is very efficient in terms of encryption and decryption time. It allows the users to be stateless and hence, they do not have to update their individual secret information with every session. It also allows dynamic revocation of users. User storage requirement is $O(\log^2 n)$ where n is the total number of users and the transmission overhead is linear in the number of revoked users r. Currently, the SD scheme offers the simplest algorithm and the best trade-offs for use in both real-time applications like Pay-TV and non-real time applications like content protection in optical discs [AAC].

1.1 Our Contributions

There are three contributions in this work.

Arbitrary number of users: We broaden the scope of use of the SD scheme. The SD scheme and all follow-up works [HS02, GST04, PB06, AK08, MMW09] assume the total number of users n to be a power of two. When implementing the SD scheme for applications such as Pay-TV, it is possible that the number of users in the system will be arbitrary. In that case, the centre has to assume the existence of dummy users to make the number of users a power of two. We relax this restriction to allow any arbitrary number of users in the system by introducing the Complete Tree Subset Difference (CTSD) scheme. The CTSD scheme is based on the SD scheme and subsumes it while eliminating the requirement of dummy users in the system. When the number of users in the CTSD method is a power of two, it becomes exactly the same as the SD scheme. Inclusion of dummy users results in the expected header length of the SD scheme to be more than the CTSD scheme for practical values of n and r.

It is to be noted that an implementation that uses the SD scheme can easily shift to using the CTSD scheme with minimal change in the software implementation. This is because the internal tree structure used for assigning keys to subsets of users in the SD scheme remains almost the same in the CTSD scheme.

Combinatorial Analysis: The importance of the SD scheme motivates the study of its combinatorial properties. We carry out such a study for the CTSD scheme and the results so obtained also apply to the SD scheme. A new approach is used for the detailed combinatorial analysis. A method is proposed to count the number, $N(n, r, h)$, of ways that r out of n users can be revoked to get a header length of h in the CTSD scheme. This counting is formulated using two recurrences. Using these recurrences, a dynamic programming based algorithm is developed to compute $N(n, r, h)$ in polynomial time. Previous to our work, to compute $N(n, r, h)$ for the SD method, one would have to run the SD algorithm on the possibly exponentially many $\binom{n}{r}$ revocation patterns. Further combinatorial results that we obtain are as follows.

1. The worst case header length for a given r in the SD scheme was shown to be $2r-1$ in [NNL01]. We show that the worst case header length for the CTSD scheme and hence for the SD scheme is $\min(2r-1, \lfloor n/2 \rfloor, n-r)$.

2. Given r, we characterize the minimum number of users, n_r, that need to be in a system using the CTSD method, that can give rise to the maximum header length of $2r-1$. For the special case of the SD method the expression for n_r was obtained in [MMW09].

3. For the special case when n is a power of two i.e., for the SD scheme, we use the recurrences to obtain a generating function for the sequence. Earlier, a generating function of a slightly different form was obtained in [PB06] using direct arguments.

Probabilistic analysis: We propose a simple and efficient algorithm for computing the expected header length for a given n and r in the CTSD and hence the SD method. The algorithm requires $O(r \log n)$ multiplications and $O(1)$ space. Due to its efficiency, this algorithm allows the computation of the expected header length for values of n ranging from a few hundreds to millions. This provides a useful tool to practitioners implementing either the SD or the CTSD method.

For the SD scheme, as n goes to infinity through powers of two, we provide an expression H_r for the limiting upper bound on the expected header length $H_{n,r}$. The value of H_r can be computed using $O(r)$ multiplications. Computing this value for different r shows that H_r is always less than $1.25r$. The only previously known upper bound on the expected header length in the SD scheme for r revoked users was proved to be $1.38r$ in [NNL01]. They also commented that experimental results indicated that the bound is probably $1.25r$. Our analysis of the expected header length shows that proving the precise limiting upper bound is more complicated than anticipated in [NNL01].

1.2 Previous Works

The tree-based SD scheme has inspired quite a lot of work in the area of broadcast encryption. Asymptotic improvements to the user storage parameter of the SD scheme were suggested in the tree-based LSD scheme of [HS02] with some loss of efficiency in the transmission overhead. Analysis of the combinatorics behind broadcast encryption schemes and different generic bounds on the efficiency parameters have been done in [LS98, PGM04] and other works. A generic method for constructing BE schemes from pseudo-random generators was proposed in [AKI03].

An analysis of the expected header length of the SD and LSD schemes was done in [PB06]. As mentioned earlier, they proposed generating functions for counting the number of ways p users out of total n users can be given access privilege so that the header length will be h. Using this generating function, they found equations to compute the expected header length for a given n and r. However, they admitted that their equations were "complex to compute and difficult to gain insight from". Consequently, they went forward to find *approximations* for the same. The analysis of the expected header length in [PB06] was continued in [EOPR08] to show that the standard deviations are small compared to the means as the number of users gets large. Other combinatorial studies of the SD method has been done in [MMW09, AK08]. In particular, the maximum possible header length for a given n and r was found accurately in [MMW09].

An earlier attempt to extend the SD method to handle arbitrary number of users have been reported in [BS11]. This work, however, considered unbalanced trees and the results so obtained are inferior to the results obtained here.

A family of broadcast encryption schemes using linear algebraic techniques and hence called linear broadcast encryption schemes was introduced in [PGMM03]. 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]. It works on the idea of "one key per punctured interval" in which the worst case header length has been brought down to r for the first time. This can also be decreased below r 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.

Traitor tracing [CFN94, FT01, NP98, KY01, SSW01] is a related issue. We do not discuss this here, since it is not directly connected to the contribution of the paper. We only remark that the traitor tracing method for the SD scheme can be modified to obtain a traitor tracing method for the CTSD scheme. There are several BE schemes based on public-key cryptography [BF99, Asa02, DF03, JG04, BGW05, GW09, LT08, PPS11]. These are not relevant to our work and we do not consider them any further.

2 The Subset Cover Revocation Framework

The Complete Tree Subset Difference method that we propose is based on the Subset Difference method introduced by Naor, Naor and Lotspiech in [NNL01]. The Subset Difference algorithm is essentially a key encrypting method that falls under the Subset Cover Revocation Framework that was proposed in the same paper. We begin with a very short description of this framework.

The Subset Cover Revocation Framework assumes a *centre* that encrypts a message M and broadcasts it to a set $\mathcal{N}$ of *users* where $|\mathcal{N}| = n$. This set of users contains all the possible recipients of the broadcast. A subset $\mathcal{R}$ of these users are revoked. A broadcast encryption algorithm under this framework consists of three parts: (1) *an initiation scheme* - that assigns user $u \in \mathcal{N}$, the secret information I_u that will allow them to decrypt messages intended for them; (2) *the broadcast algorithm* - that takes as input the message M and the set $\mathcal{R}$ of revoked users and outputs the ciphertext C. C is broadcast to all the users in $\mathcal{N}$; (3) *the decryption algorithm* - that runs at the user end. It takes as input the ciphertext C and the secret information I_u that the user u had received during initiation and attempts to decrypt C. A privileged user in $\mathcal{N} \setminus \mathcal{R}$ should be able to get back the original message M, while any coalition of revoked users in $\mathcal{R}$ should not be able to get back the correct message from C.

During initiation, the algorithm defines a collection $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_w\}$ of subsets, where each $\mathcal{S}_j \subseteq \mathcal{N}$. Each subset $\mathcal{S}_j$ is assigned a long-lived key L_j. A user $u \in \mathcal{S}_j$ should be able to deduce L_j from the secret I_u it had acquired during initiation. However, I_u may not explicitly contain the long-lived key L_j, as we will see in the Complete Tree Subset Difference algorithm. During broadcast, the set of privileged users $\mathcal{N} \setminus \mathcal{R}$ is partitioned into pairwise disjoint subsets $\mathcal{S}_{i_1}, \ldots, \mathcal{S}_{i_h}$ taken from the collection $\mathcal{S}$. This partition is called the *subset cover* $\mathcal{S}_c$. In other words,

$$\mathcal{N} \setminus \mathcal{R} = \bigcup_{j=1}^{h} \mathcal{S}_{i_j}$$

where each $\mathcal{S}_{i_j} \in \mathcal{S}$ and $\mathcal{S}_c = \{\mathcal{S}_{i_1}, \ldots, \mathcal{S}_{i_h}\}$. The algorithm uses two encryption schemes:

- A function $F_K : \{0,1\}^* \rightarrow \{0,1\}^*$ to encrypt the message M with a *session key* K. The session key is a random string chosen afresh for each new message M.
- A function $E_{L_j} : \{0,1\}^* \rightarrow \{0,1\}^*$ to encrypt the session key K with a long-lived key L_j corresponding to the subset $\mathcal{S}_j$ $(\in \mathcal{S}_c)$ of users.

Detailed discussion on the security requirement of these primitives can be found in [NNL01]. Hence, in order to broadcast the message M, the centre chooses a session key K and encrypts M as $F_K(M)$. It also finds the cover $\mathcal{S}_c = \{\mathcal{S}_{i_1}, \ldots, \mathcal{S}_{i_h}\}$. Let $L_{i_1}, \ldots, L_{i_h}$ be the long-lived keys that were assigned to each of these subsets in $\mathcal{S}_c$. The centre then encrypts the session key K with each of these keys L_{i_j}. The session key has to be encrypted h times for each set in $\mathcal{S}_c$. The h encryptions of the session key is sent along with $F_K(M)$ as a *header* for the encrypted message. The header also has information to identify the subsets $\mathcal{S}_{i_j}$ that form the cover $\mathcal{S}_c$. The size h of the header is determined by the number of sets in $\mathcal{S}_c$. We are going to refer to this size as the *header length*. The encrypted message $F_K(M)$ along with the header forms the ciphertext C. The header length is a key efficiency parameter that resembles the transmission overhead of the scheme.

During decryption, a user u has to identify from the header, the set $\mathcal{S}_{i_j}$ to which it belongs. It derives the long-lived key L_{i_j} from the secret information I_u it had acquired during initiation. Using L_{i_j}, it then decrypts the session key K from the portion of the header that has K encrypted for $\mathcal{S}_{i_j}$. The user can hence decrypt the message M from $F_K(M)$. In case a user is revoked and hence does not belong to any of the sets in $\mathcal{S}_c$, it will not be able to decrypt K or M for that matter.

3 The Complete Tree Subset Difference Method

The Subset Difference (SD) method of [NNL01] and all follow-up work assumes the number of users n to be a power of two. We propose the Complete Tree Subset Difference (CTSD) algorithm that can accommodate any arbitrary number of users. Our algorithm considers a rooted complete binary tree $\mathcal{T}^0$ with n leaves. One may

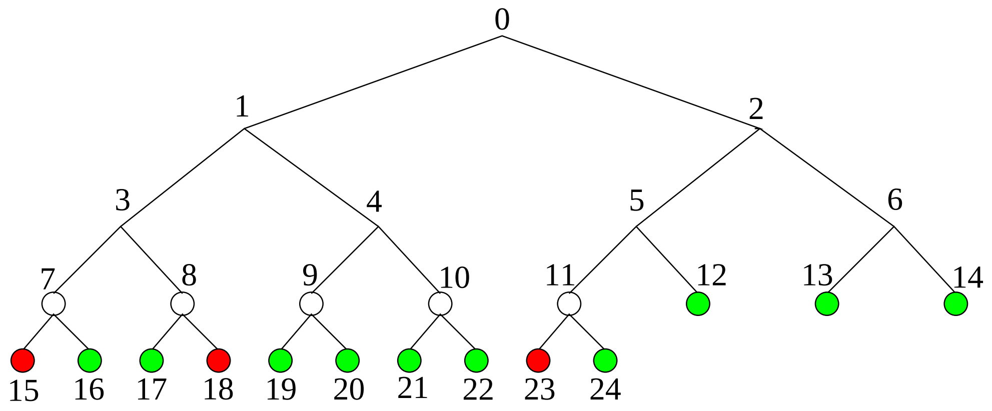

Figure 1: The *non-full complete* tree $\mathcal{T}^0$ with $n = 13$ users as its leaves. Privileged users are indicated in green and the revoked users are indicated in red. Here, $r = 3$. The tree $\mathcal{T}^1$ is a subtree of $\mathcal{T}^0$ and is a *full* subtree having 8 leaf nodes whereas the tree $\mathcal{T}^2$ is a *non-full complete* subtree of $\mathcal{T}^0$ with 5 leaf nodes.

note here that a *complete binary tree* has leaf nodes only at the bottom-most or last level and maybe also the last-but-one level. The leaves in the last level are filled from the left to the right in the tree. In a *full binary tree* of height ℓ there are 2^ℓ leaves, all at the last level. A full binary tree is also complete by definition. We will refer to trees that are complete but not full as *non-full*. Each user in $\mathcal{N}$ is associated with a leaf of the complete binary tree $\mathcal{T}^0$. There are a total of $2n-1$ nodes in $\mathcal{T}^0$. The root node of $\mathcal{T}^0$ is labeled as 0. All subsequent nodes are labeled as follows: the left child node of a node i is labeled as $2i+1$ and the right child is labeled as $2i+2$. Hence, nodes 0 to $n-2$ are the internal nodes and nodes $n-1$ to $2n-2$ are the leaf nodes. The subtree of $\mathcal{T}^0$ rooted at node i is denoted by $\mathcal{T}^i$. The number of leaf nodes in the subtree $\mathcal{T}^i$ is denoted by λ_i. The collection $\mathcal{S}$ of subsets is defined as follows: The set $S_{i,j}$ is defined to contain users in the subtree $\mathcal{T}^i$ but *not* in $\mathcal{T}^j$. A set $S_{i,j}$ is also denoted as $\mathcal{T}^i \setminus \mathcal{T}^j$. All subsets of users of the form $S_{i,j}$, where node j is in the subtree $\mathcal{T}^i$ and hence a *descendant* of node i, is included in the collection $\mathcal{S}$. The set $\mathcal{N}$ of all users is also included in $\mathcal{S}$.

Once this collection $\mathcal{S}$ has been created, each set $S_{i,j}$ in $\mathcal{S}$ has to be assigned a long-lived key $L_{i,j}$. We will look at the key assignment in Section 3.1.

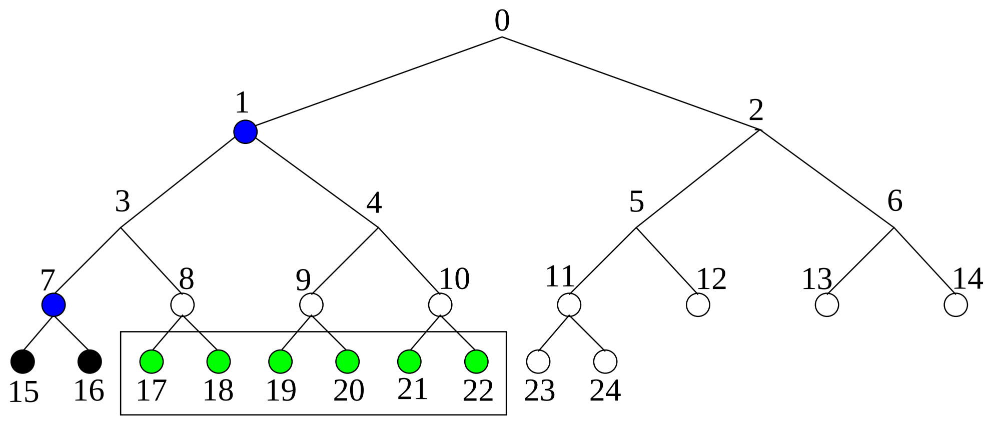

Figure 2: The *subset difference* subset $S_{1,7}$ which includes leaves in $\mathcal{T}^1$ but not in $\mathcal{T}^7$ i.e.; $S_{1,7} = \mathcal{T}^1 \setminus \mathcal{T}^7 = \{17, 18, 19, 20, 21, 22\}$.

During broadcast, the centre will know the set $\mathcal{R}$ of revoked users and the message M to be broadcast. It has to find the subset cover $\mathcal{S}_c$ for $\mathcal{N} \setminus \mathcal{R}$. $\mathcal{S}_c$ contains pairwise disjoint sets $S_{i_1,j_1}, \ldots, S_{i_h,j_h}$ such that $\mathcal{N} \setminus \mathcal{R} = \bigcup_{k=1}^{h} \mathcal{S}_{i_k,j_k}$ where each S_{i_k,j_k} is taken from $\mathcal{S}$. If the set $\mathcal{R}$ is empty, then the only set in the cover $\mathcal{S}_c$ is $\mathcal{N}$. Otherwise, the following *cover-finding algorithm* is used: The centre first constructs the Steiner Tree $\mathcal{ST}(\mathcal{R})$ induced by $\mathcal{R}$ on $\mathcal{T}^0$. The Steiner Tree $\mathcal{ST}(\mathcal{R})$ is a subgraph of $\mathcal{T}^0$ that only retains the nodes and edges on paths from the root node 0 to a revoked leaf node. All the other paths in $\mathcal{T}^0$ are deleted. The cover-finding algorithm runs iteratively by maintaining a tree $\mathcal{T}$ that is a sub-graph of $\mathcal{ST}(\mathcal{R})$. It starts by initializing $\mathcal{T}$ as a copy of $\mathcal{ST}(\mathcal{R})$. At every iteration, the algorithm keeps removing nodes from $\mathcal{T}$ while adding subsets to $\mathcal{S}_c$, until $\mathcal{T}$ has just one node left. At any point of time in the algorithm, a leaf node in $\mathcal{T}$ corresponds to either a leaf node in $\mathcal{T}^0$ or the root of a subtree in $\mathcal{T}^0$ all whose leaves have already been covered till that iteration. More precisely:

1. If there is only one leaf node in $\mathcal{T}$, jump to step 6.
2. Find two leaves j_1 and j_2 of $\mathcal{T}$ whose first common *ancestor* i does not have any other leaf node in its subtree in $\mathcal{T}$. Here, out of the many possible such pairs j_1 and j_2 one may choose the leftmost to have a specific algorithm.
3. Let i_1 (respectively i_2) be the immediate child node of i which is an ancestor of j_1 (respectively j_2) or is the node j_1 (respectively j_2) itself. If $i_1 \neq j_1$ then add the set S_{i_1,j_1} to the cover $\mathcal{S}_c$. Similarly, if $i_2 \neq j_2$ then add the set S_{i_2,j_2} to the cover $\mathcal{S}_c$.
4. Delete the paths joining j_1 and j_2 with their common ancestor i. Hence, node i becomes a leaf in $\mathcal{T}$.
5. If there are more than one leaves remaining in $\mathcal{T}$, go back to step 2.
6. If the only leaf node is the node 0, then there are no more subsets to be added to $\mathcal{S}_c$. Else, add the set $S_{0,j}$ to $\mathcal{S}_c$. Here j is the leaf node remaining in $\mathcal{T}$.

3.1 Key assignment to each subset $S_{i,j}$ in $\mathcal{S}$

Pseudo-random generator G: In order to assign keys to each subset in $\mathcal{S}$, the centre assigns uniform random seeds to every non-leaf node in $\mathcal{T}^0$ and uses a *cryptographic pseudo-random generator* G. The pseudo-random generator G outputs a pseudo-random string that has three times the length of the input seed. The output string $G(seed)$ is divided into three equal parts $G_L(seed)$, $G_M(seed)$ and $G_R(seed)$. Hence, $G(seed) = G_L(seed) \parallel G_M(seed) \parallel G_R(seed)$. $G : \{0,1\}^k \to \{0,1\}^{3k}$ is a pseudo-random generator if no polynomial time adversary can distinguish between its output for a random seed from a truly random string of the same length.

Seed assignment to nodes: Every non-leaf node i in $\mathcal{T}^0$ is assigned a uniform random seed $LABEL_i$. Each non-root node j of $\mathcal{T}^0$ is assigned derived seeds from every ancestor i of j. The left child $2i+1$ of node i in $\mathcal{T}^0$ derives the seed $G_L(LABEL_i)$ from the random seed $LABEL_i$ of i. All descendants of $2i+1$ further get derived seeds from this derived seed $G_L(LABEL_i)$ of $2i+1$. Similarly, the right child $2i+2$ of node i in $\mathcal{T}^0$ derives the seed $G_R(LABEL_i)$ from the random seed of i and all descendants of $2i+2$ get derived seeds from this derived seed $G_R(LABEL_i)$ of $2i+2$. We denote the seed for a node j derived from the random seed of node i as $LABEL_{i,j}$. Following such an assignment of random and derived seeds for nodes in $\mathcal{T}^0$, the long lived key $L_{i,j}$ assigned to the set $S_{i,j}$ is $G_M(LABEL_{i,j})$.

I_u for each $u \in \mathcal{N}$: Once the centre is done with the assignment of random and derived seeds to nodes, it has to distribute the secret information I_u to each user $u \in \mathcal{N}$. The user associated with a leaf j of $\mathcal{T}^0$ must have been revoked when a set $S_{i,j}$ is in the cover $\mathcal{S}_c$. Hence, the user at leaf j should not be able to compute the $L_{i,j}$ for any of its predecessor i in $\mathcal{T}^0$. In fact, it should not be able to compute any $L_{i,k}$ where k, a descendant of i, is also one of its ancestors. In other words, a user at leaf j should be able to compute an $L_{i,k}$ if and only if i is an ancestor of j and k being a descendant of i, is not on the path joining j with i. In a subtree $\mathcal{T}^i$ of $\mathcal{T}^0$ to which a user at leaf j belongs, the node i has a random seed $LABEL_i$. The user at j gets the seeds of all nodes adjacent to the path joining i and j that have been derived from $LABEL_i$. Say $i_1, \ldots, i_m$ are those nodes "falling off" from the path between node i and leaf j. The user at j will get the derived seeds $LABEL_{i,i_1}, LABEL_{i,i_2}, \ldots, LABEL_{i,i_m}$. To summarize, the I_u for a user u at leaf j consists of all derived seeds $LABEL_{i,k}$ such that i is a predecessor of j and k is adjacent to the path joining i and j. As derived in [NNL01], the number of derived seeds in I_u is $\frac{1}{2}\log^2 n + \frac{1}{2}\log n + 1$ for n a power of two. For an arbitrary n, one has to consider the next higher power of two, say $2^{\ell_0 - 1} < n \leq 2^{\ell_0}$. The number of derived seeds in I_u will be $\frac{1}{2}\ell_0^2 + \frac{1}{2}\ell_0 + 1$.

3.2 Dummy Users and the Associated Penalty

The CTSD scheme works with the actual number of users that are present in the system. It may be argued that even if n is not a power of two, the SD scheme can be applied by incorporating dummy users to make the total number of users to be a power of two. We argue that this impacts the size of the transmission overhead. For an actual broadcast, there are two ways to handle the dummy users – either consider all of them to be revoked or consider all of them to be privileged.

Suppose that the dummy users are considered to be distributed randomly among all the users. Then viewing them as revoked has very serious performance penalties. This is because, the average header length is linear in the number of revoked users, as is proved later. Having a larger number of *randomly distributed* revoked users leads to larger header size. If, on the other hand, the dummy users are viewed as privileged, then the performance penalty will be lesser.

Assuming the dummy users to be randomly distributed may not be fully justifiable. In an actual implementation, they may be considered to be one block. Suppose that $2^{\ell-1} < n < 2^\ell$ and that the users numbered $n+1, \ldots, 2^\ell$ are the dummy users and the real users are numbered 1 to n. The actual revoked users will be

n	$r = 2$	$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$	$r = 8$
17 (CTSD)	2.34	3.22	3.93	4.49	4.89	5.13	5.21
$17 + 15$ (dummy revoked)	3.06	3.87	4.49	4.96	5.29	5.46	5.49
$17 + 15$ (dummy privileged)	2.76	3.88	4.66	5.24	5.64	5.87	5.96
18 (CTSD)	2.36	3.29	4.05	4.67	5.14	5.45	5.60
$18 + 14$ (dummy revoked)	3.04	3.88	4.53	5.04	5.41	5.65	5.74
$18 + 14$ (dummy privileged)	2.67	3.76	4.53	5.09	5.51	5.78	5.92
19 (CTSD)	2.37	3.32	4.09	4.73	5.21	5.55	5.74
$19 + 13$ (dummy revoked)	3.12	4.01	4.72	5.27	5.69	5.97	6.11
$19 + 13$ (dummy privileged)	2.61	3.72	4.52	5.16	5.67	6.07	6.35
20 (CTSD)	2.39	3.38	4.19	4.86	5.39	5.77	6.02
$20 + 12$ (dummy revoked)	2.86	3.70	4.40	4.98	5.44	5.80	6.03
$20 + 12$ (dummy privileged)	2.56	3.66	4.48	5.15	5.69	6.12	6.44
21 (CTSD)	2.40	3.38	4.20	4.88	5.43	5.85	6.15
$21 + 11$ (dummy revoked)	3.69	4.44	5.07	5.60	6.02	6.35	6.56
$21 + 11$ (dummy privileged)	2.52	3.64	4.52	5.26	5.90	6.43	6.84
22 (CTSD)	2.42	3.43	4.27	4.98	5.58	6.06	6.42
$22 + 10$ (dummy revoked)	3.19	4.09	4.86	5.50	6.01	6.40	6.69
$22 + 10$ (dummy privileged)	2.49	3.62	4.53	5.31	5.99	6.56	7.03
23 (CTSD)	2.43	3.44	4.28	4.99	5.60	6.09	6.48
$23 + 9$ (dummy revoked)	3.27	4.20	5.01	5.68	6.23	6.66	6.98
$23 + 9$ (dummy privileged)	2.47	3.62	4.58	5.41	6.14	6.77	7.28
24 (CTSD)	2.45	3.48	4.33	5.07	5.71	6.24	6.67
$24 + 8$ (dummy revoked)	2.70	3.54	4.35	5.08	5.71	6.24	6.67
$24 + 8$ (dummy privileged)	2.45	3.60	4.59	5.45	6.19	6.83	7.34

Table 1: Comparison of the expected header lengths for $17 \leq n \leq 24$ and $2 \leq r \leq 8$ in the CTSD method with the SD method working with dummy users forming a block at the right end. The dummy users may be privileged or revoked. It shows that the CTSD scheme always requires lesser bandwidth than the SD scheme with dummy users.

among the values 1 to n, whereas the users numbered $n + 1, \ldots, 2^{\ell}$ will be considered to be either all revoked or all privileged.

We compare the expected header length of the CTSD method with the SD method in Table 1. These values are obtained by running the header generation algorithms on all possible (n, r)-revocation patterns. The SD algorithm is run assuming the dummy users to form a block at the right end of the tree. In separate cases, these dummy users are considered to be privileged and revoked as a group. Due to the exponentially many possible revocation patterns, the algorithm could be run only for small values of n. We, however, expect the results to be indicative of the general behaviour. For $17 \leq n \leq 24$ and $2 \leq r \leq 8$, the expected header length by the CTSD method is never more than that of the SD method and is almost always less.

4 Combinatorial Analysis of the SD and CTSD methods

A given set of revoked users is called a *revocation pattern*. We denote a revocation pattern on n users where r are revoked, as an (n, r)-*revocation pattern*. The number of possible (n, r)-revocation patterns is $\binom{n}{r}$. In order to study the detailed combinatorial behaviour of the CTSD and hence the SD algorithm, we find a method to count the number of (n, r)-revocation patterns that result in a header length of h.

Definition 1. *In a subtree $\mathcal{T}^j$ of $\mathcal{T}^0$ with λ_j users, $N(\lambda_j, r, h)$ is defined as the number of (λ_j, r)-revocation patterns that are covered by exactly h subsets. Similarly, for λ_j users in $\mathcal{T}^j$, $T(\lambda_j, r, h)$ is defined as the number of (λ_j, r)-revocation patterns that are covered by h subsets such that there is at least one revoked user in both subtrees of $\mathcal{T}^j$.*

Since the tree $\mathcal{T}^0$ has n $(= \lambda_0)$ leaves, $N(n, r, h) = N(\lambda_0, r, h)$ is the number of (n, r)-revocation patterns covered by a header length of h. We obtain recurrences for $N(n, r, h)$.

4.1 Some Notation

Level number and position of nodes: Before we start deriving the expressions for $T(n, r, h)$ and $N(n, r, h)$, we fix a few notation for the ease of description. A level number of $\mathcal{T}^0$ is indicated by ℓ. In particular, the level of a node i is denoted by ℓ_i. The root node 0 is at the highest level ℓ_0. Hence, $\ell \in \{0, \dots, \ell_0\}$. Since every subtree $\mathcal{T}^i$ is a complete binary tree, $2^{\ell_i - 1} < \lambda_i \leq 2^{\ell_i}$. The number of nodes at level ℓ of $\mathcal{T}^0$ is denoted by q_ℓ. We see that the number of nodes at the last level is $q_0 = 2(n - 2^{\ell_1})$. For $\ell \in \{1, \dots, \ell_0\}$, $q_\ell = 2^{\ell_0 - \ell}$. The position of a node at a level from the left is denoted by k where k ranges from 1 to q_ℓ. Hence, a node i is uniquely represented by the pair (ℓ_i, k_i) – the level ℓ_i of $\mathcal{T}^0$ to which it belongs and its position k_i from the left at that level. As an example, the root node 0 of $\mathcal{T}^0$ is represented by $(\ell_0, 1)$. We will interchangeably use both i and (ℓ_i, k_i) to denote a node.

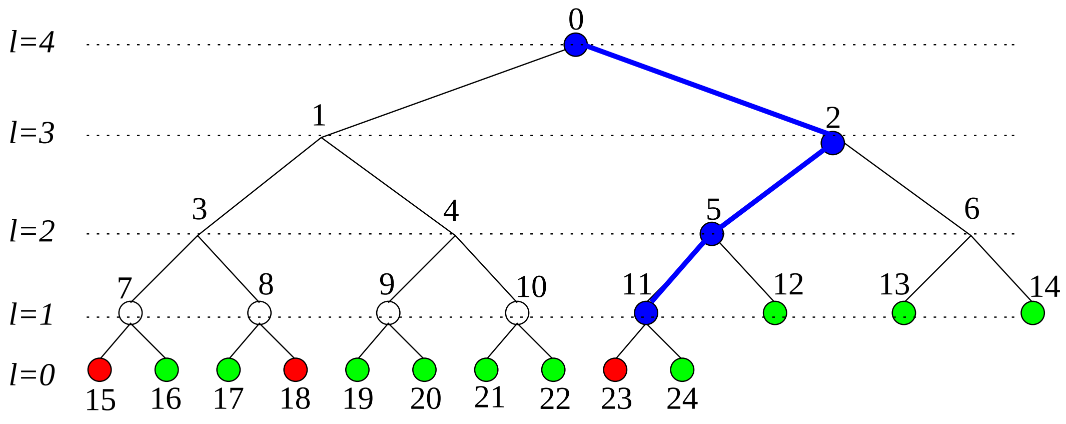

Figure 3: Level numbers in $\mathcal{T}^0$. The path $\mathcal{P}^0$ is marked with blue. Nodes coloured blue are at position $k_\ell^{\mathcal{P}}$ for the respective level ℓ.

Non-full subtrees at each level of $\mathcal{T}^0$: Let us take a closer look at the structure of the tree $\mathcal{T}^0$. In case $\mathcal{T}^0$ is full, all its subtrees are also full. In case $\mathcal{T}^0$ is non-full, we observe that every level $\ell > 0$ of $\mathcal{T}^0$ can have at most one non-full subtree. To identify these subtrees, we look at the path joining the root node 0 of $\mathcal{T}^0$ with node $n - 2$ and denote it by $\mathcal{P}_0$. The node numbered $n - 2$ is the last non-leaf node. There is exactly one node on $\mathcal{P}_0$ for every level $\ell > 0$ of $\mathcal{T}^0$. For level ℓ, the position of the node lying on the path $\mathcal{P}_0$ from the left, is denoted by $k_\ell^{\mathcal{P}}$. Let j be a node on $\mathcal{P}_0$, say the node represented by $(\ell, k_\ell^{\mathcal{P}})$. The part of the path $\mathcal{P}_0$ lying in the subtree $\mathcal{T}^j$ is denoted as $\mathcal{P}_j$. For the level ℓ, the subtree $\mathcal{T}^j$ rooted at node $(\ell, k_\ell^{\mathcal{P}})$ is the only possibly non-full subtree rooted at level ℓ. The subtrees to the left and right of node $k_\ell^{\mathcal{P}}$ at level ℓ are all full. The subtrees to the left (respectively right) of node $k_\ell^{\mathcal{P}}$ of level ℓ have 2^ℓ (respectively $2^{\ell-1}$) leaves. The number of leaves in the only possibly non-full subtree rooted at level ℓ is denoted by $\lambda^{\ell,\mathcal{P}}$. The root node of this subtree would be node $(\ell, k_\ell^{\mathcal{P}})$ of level ℓ. Hence, $2^{\ell-1} < \lambda^{\ell,\mathcal{P}} \leq 2^\ell$. More specifically, $\lambda^{\ell,\mathcal{P}} = n - ((k_\ell^{\mathcal{P}} - 1) \times 2^\ell) - ((2^{\ell_0-\ell} - k_\ell^{\mathcal{P}}) \times 2^{\ell-1})$. Also, $k_\ell^{\mathcal{P}} = \lceil \frac{q_0}{2^\ell} \rceil$. We define $k_\ell^{\mathcal{P}_j}$ for the path $\mathcal{P}_j$ as the position of the node at level ℓ on $\mathcal{P}_j$ from the left in the subtree $\mathcal{T}^j$. Hence, $k_\ell^{\mathcal{P}}$ is also denoted as $k_\ell^{\mathcal{P}_0}$. One can see that $k_\ell^{\mathcal{P}_j} = \left\lceil \frac{q_0 - (k_{\ell_j}^{\mathcal{P}} - 1) \times (2^{\ell_j})}{2^\ell} \right\rceil = \lceil \frac{q_0}{2^\ell} \rceil - (k_{\ell_j}^{\mathcal{P}} - 1) \times (2^{\ell_j - \ell})$.

4.2 Recurrences $N(n, r, h)$ and $T(n, r, h)$

Theorem 1. *For a subtree $\mathcal{T}^i$ of $\mathcal{T}^0$ with λ_i ($2^\ell < \lambda_i \leq 2^{\ell+1}$) leaves,*

$$N(\lambda_i, r_1, h_1) = T(\lambda_i, r_1, h_1) + \sum_{j \in \mathsf{IN}(i)} T(\lambda_j, r_1, h_1 - 1) \tag{1}$$

where $\mathsf{IN}(i)$ is the set of all internal nodes in the subtree $\mathcal{T}^i$ excluding the node i.

Proof. We show that a revocation pattern is counted in $N(\lambda_i, r_1, h_1)$ if and only if it is counted in exactly one of $T(\lambda_i, r, h)$ or $T(\lambda_j, r, h-1)$ for some $j \in \mathsf{IN}(i)$. First we consider a (λ_i, r)-revocation pattern that is counted in $N(\lambda_i, r, h)$. There exists a minimal subtree $\mathcal{T}^j$, with $j \in \mathsf{IN}(i)$, of $\mathcal{T}^i$ that contains all the revoked leaves. If this subtree is rooted at i itself, then that revocation pattern is counted in $T(\lambda_i, r, h)$ and is covered by h subsets of $\mathcal{S}$. For any other node $j \neq i$, the revocation pattern is counted in $T(\lambda_j, r, h-1)$ and has to be covered by $h-1$ subsets of $\mathcal{S}$. The rest of the $\lambda_i - \lambda_j$ privileged users form one SD subset of the cover. The total cover size will hence be h. Since a set $\mathcal{R}$ of revoked users has a corresponding unique minimal subtree $\mathcal{T}^j$ of $\mathcal{T}^i$ containing all the users in $\mathcal{R}$, hence it is counted exactly once on the right side of (1).

Now, let us consider a (λ_i, r)-revocation pattern that has been counted in $T(\lambda_i, r, h)$. By the definitions of T and N, the (λ_i, r)-revocation patterns that are counted in $T(\lambda_i, r, h)$ are also counted in $N(\lambda_i, r, h)$. For some other revocation pattern, counted in $T(\lambda_j, r, h-1)$ for some $j \in \mathsf{IN}(i)$, both subtrees of $\mathcal{T}^j$ contain at least one revoked user in each. Hence, the minimal subtree of $\mathcal{T}^i$ containing the r revoked users for such a revocation pattern is $\mathcal{T}^j$. For the revocation patterns counted in $T(\lambda_j, r, h-1)$, the privileged users of the subtree $\mathcal{T}^j$ have been covered with $h-1$ SD subsets of $\mathcal{S}$. The rest of the $\lambda_i - \lambda_j$ users are all privileged and are covered by one more SD subset $S_{i,j}$. Hence, the corresponding (λ_i, r)-revocation pattern is counted in $N(\lambda_i, r, h)$. □

Theorem 2. *For a subtree $\mathcal{T}^i$ of $\mathcal{T}^0$ with λ_i ($2^\ell < \lambda_i \leq 2^{\ell+1}$) leaves,*

$$T(\lambda_i, r_1, h_1) = \sum_{r'=1}^{r_1-1} \sum_{h'=0}^{h_1} N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h') \tag{2}$$

where λ_{2i+1} (respectively λ_{2i+2}) is the number of leaves in the left (respectively right) subtree of $\mathcal{T}^i$.

Proof. We show that a revocation pattern is counted in $T(\lambda_i, r_1, h_1)$ if and only if it is counted in the right hand side of (2). For a given λ_i, the number of leaves in the left and right subtrees get fixed to λ_{2i+1} and λ_{2i+2} respectively. When a (λ_i, r_1)-revocation pattern is counted in $T(\lambda_i, r_1, h_1)$, both the subtrees of $\mathcal{T}^i$ must have at least one revoked user. Assuming the left subtree of $\mathcal{T}^i$ has r' revoked users, the right subtree should have $r_1 - r'$ revoked users since the total number of revoked users is r_1. Similarly, assuming that the privileged users in this left subtree are covered by h' sets of $\mathcal{S}$, the privileged users in the right subtree should be covered by $h_1 - h'$ sets of $\mathcal{S}$. The number of (λ_{2i+1}, r')-revocation patterns in the left subtree covered by h' subsets

$T(\lambda_i, r_1, h_1)$	$r_1 < 0$	$r_1 = 0$	$r_1 = 1$	$2 \le r_1 < n$	$r_1 = n$	$r_1 > n$
$h_1 = 0$	0	0	0	0	1	0
$h_1 \ge 1$	0	0	0	from (2)	0	0

$N(\lambda_i, r_1, h_1)$	$r_1 < 0$	$r_1 = 0$	$r_1 = 1$	$2 \le r_1 < n$	$r_1 = n$	$r_1 > n$
$h_1 = 0$	0	0	0	0	1	0
$h_1 = 1$	0	1	n	from (1)	0	0
$h_1 > 1$	0	0	0	from (1)	0	0

Table 2: Boundary conditions on $T(n, r, h)$ and $N(n, r, h)$.

is $N(\lambda_{2i+1}, r', h')$. Similarly, the number of $(\lambda_{2i+2}, r_1 - r')$-revocation patterns in the right subtree covered by $h_1 - h'$ subsets is $N(\lambda_{2i+2}, r_1 - r', h_1 - h')$. Each such (λ_{2i+1}, r')-revocation pattern in the left subtree along with a $(\lambda_{2i+2}, r_1 - r')$-revocation pattern in the right subtree gives rise to a (λ_i, r)-revocation pattern in the tree $\mathcal{T}^i$ that is covered by h_1 subsets of $\mathcal{S}$. Hence, for all values of $r' \in \{1, \ldots, r_1 - 1\}$ and all values of $h' \in \{0, \ldots, h_1\}$, $N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$ counts all the possible $T(\lambda_i, r_1, h_1)$.

Any (λ_i, r_1)-revocation pattern covered by h' subsets will be counted in some $N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$. The ones counted in $N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$ for fixed values of r' and h' are counted exactly once in it. For other values of r' and h', the corresponding (λ_i, r_1)-revocation patterns will be counted in the respective $N(\lambda_{2i+1}, r', h') \times N(\lambda_{2i+2}, r_1 - r', h_1 - h')$. Hence, a (λ_i, r_1)-revocation pattern is counted on the right hand side of (2) if and only if it is counted in $T(\lambda_i, r_1, h_1)$. □

Boundary conditions: The boundary conditions on $T(\lambda_i, r_1, h_1)$ and $N(\lambda_i, r_1, h_1)$ are given in Table 2. Other than the tabulated values, $N(\lambda_i, r_1, h_1) = 0$ for $\lambda_i \le 0$ and $T(\lambda_i, r_1, h_1) = 0$ for $\lambda_i \le 1$. From recurrences in Theorems 1 and 2 and the boundary conditions on these recurrences, one can find the value of $N(n, r, h)$ for any given n, r and h using dynamic programming.

4.3 Algorithms to compute $N(n, r, h)$ and $T(n, r, h)$

Substituting for $j \in \mathsf{IN}(i)$: To use these recurrences as an algorithm, the nodes $j \in \mathsf{IN}(i)$ in (1) for a node i have to be explicitly identified and the corresponding λ_js have to be substituted. As described in Section 4.1 before, there are at most three types of subtrees rooted at a level ℓ_j of $\mathcal{T}^0$: full subtrees of height ℓ_i, full subtrees of height $\ell_i - 1$ and a non-full complete subtree of height ℓ_i.
(1) For a subtree $\mathcal{T}^i$ that is full and is of height 2^{ℓ_i} and to the left of the node at position $k^{\mathcal{P}}_{\ell_i}$ at level ℓ_i:

$$N(\lambda_i, r_1, h_1) = T(\lambda_i, r_1, h_1) + \sum_{\ell_j=1}^{\ell_i - 1} (2^{\ell_i - \ell_j}) \times T(2^{\ell_j}, r_1, h_1 - 1). \tag{3}$$

(2) For a subtree $\mathcal{T}^i$ that is full and is of height $2^{\ell_i - 1}$ and to the right of the node at position $k^{\mathcal{P}}_{\ell_i}$ at level ℓ_i:

$$N(\lambda_i, r_1, h_1) = T(\lambda_i, r_1, h_1) + \sum_{\ell_j=2}^{\ell_i - 1} (2^{\ell_i - \ell_j}) \times T(2^{\ell_j - 1}, r_1, h_1 - 1). \tag{4}$$

(3) For the only possibly non-full subtree $\mathcal{T}^i$ for $i = (\ell_i, k^{\mathcal{P}}_{\ell_i})$ of height 2^{ℓ_i} and at position $k^{\mathcal{P}}_{\ell_i}$ at level ℓ_i:

$$\begin{aligned} N(\lambda_i, r_1, h_1) = T(\lambda_i, r_1, h_1) & \\ &+ \sum_{\ell_j=2}^{\ell_i-1} [(k^{\mathcal{P}_i}_{\ell_j} - 1) \times T(2^{\ell_j}, r_1, h_1 - 1) + T(\lambda^{\ell_j,\mathcal{P}}, r_1, h_1 - 1) \\ &\qquad + (2^{\ell_i - \ell_j} - k^{\mathcal{P}_i}_{\ell_j}) \times T(2^{\ell_j - 1}, r_1, h_1 - 1)]. \end{aligned} \tag{5}$$

Dynamic Programming: Computing $N(n, r, h)$ and $T(n, r, h)$ requires computing the values of $N(\lambda_i, r_1, h_1)$ and $T(\lambda_i, r_1, h_1)$ for some smaller λ_i, r_1 and h_1. We use dynamic programming technique where all values of $N(\lambda_i, r_1, h_1)$ and $T(\lambda_i, r_1, h_1)$ for smaller λ_i, r_1 and h_1 are pre-computed. The algorithm to compute $T(n, r, h)$ from these pre-computed values is obtained from (2) in a straightforward manner. The algorithm to compute $N(n, r, h)$ from these pre-computed values is obtained from (1). More specifically from either of (3) or (5). Level ℓ_i of $\mathcal{T}^0$ has $k^{\mathcal{P}}_{\ell_i} - 1$ full subtrees of height ℓ_i, $(2^{\ell_0 - \ell_i}) - k^{\mathcal{P}}_{\ell_i}$ full subtrees of height $\ell_i - 1$ and one possibly non-full subtree. For every level in the tree $\mathcal{T}^0$, $T(\lambda_i, r, h - 1)$ is pre-computed once for each of the three types of nodes and used to compute $N(n, r, h)$.

Space and Time complexity of the algorithm: Using (2) to compute $T(n, r, h)$ from the pre-computed values of $N(\cdot, \cdot, \cdot)$ requires $O(rh)$ memory operations and multiplications. Equation (1) shows how $N(n, r, h)$ is related to pre-computed values of $T(\cdot, \cdot, \cdot)$. Actual computation is done using (3), (4) and (5). This requires $O(1)$ memory operations and a single addition for each of the $\lceil \log n \rceil$ levels of $\mathcal{T}^0$. Hence, the time complexity for computing $T(n, r, h)$ and then $N(n, r, h)$ *from pre-computed values* is $O(rh + \log n)$.

These pre-computed values in turn need to be computed. By the form of (3), (4) and (5) there are $\log n$ subtrees to be considered. For each such subtree, $O(rh)$ values need to be computed and the computation of these will be based on values computed earlier. A dynamic programming algorithm proceeds in a bottom-up fashion by computing the $O(rh)$ values corresponding to smaller sub-trees and then using these to compute the values for progressively larger sub-trees. This takes a total of $O(r^2 h^2 \log n + rh \log^2 n)$ time. The space requirement is given by the number of pre-computed values that need to be stored to compute $N(n, r, h)$. For each of the $O(\log n)$ sub-trees, a total of $O(rh)$ values need to be stored and so the space complexity is $O(rh \log n)$.

The above time and space complexities are required for a single set of values of n, r and h. For a fixed n and r, it may be required to compute the values of $N(n, r, h)$ for all possible values of h. This would be a typical requirement for a broadcast centre which will have a fixed number of users and for a particular transmission knows the number of revoked users. The corresponding time and space complexities can be obtained by substituting an appropriate value for h. In Lemma 9 in Appendix A, we show that $h \leq 2r - 1$ which gives the expressions $O(r^4 \log n + r^2 \log n)$ and $O(r^2 \log^2 n)$ for time and space complexities respectively. For large n and moderate values of r, these are practical complexities.

Further, allowing r to range over all the $O(n)$ possible values leads to $O(n^4 \log n + n^2 \log^2 n)$ time and $O(n^2 \log n)$ space complexities respectively. If we are interested in computing $N(i, r, h)$ for all $2 \leq i \leq n$ and all possible values of r and h, then the time and space complexities are $O(n^5 + n^3 \log n)$ and $O(n^3)$ respectively.

As an example, using this dynamic programming algorithm, we find that for $n = 126$, $r = 63$ and $h = 37$, the floating point value of $N(n, r, h)$ is 7.44×10^{35}. Note that computing such a value would not be possible by direct enumeration. Attempting direct enumeration, would require considering $\binom{126}{63}$ possible revocation patterns which is way beyond the present computational capabilities.

4.4 Upper Bounds on the Header Length

The header length is an important efficiency parameter of a broadcast encryption scheme. So, upper bounds on the header length of the SD and CTSD schemes are of practical interest. A detailed combinatorial analysis of upper bounds on the header length is given in Appendix A. Here, we present a summary of the important results. The proofs are given in Appendix A.

In Lemma 9 in Appendix A, it is shown that the header length of the CTSD scheme is at most $2r-1$. For the special case of the SD method, this bound was proved in [NNL01]. This bound is made more specific in Theorem 3 below for the CTSD and hence the SD method.

Theorem 3. *The maximum header length in the CTSD method for n users is* $\min(2r-1, \lfloor \frac{n}{2} \rfloor, n-r)$.

The bound given by Theorem 3 gives a complete picture. If $r \le n/4$, then the bound $2r-1$ is appropriate; if $n/4 < r \le n/2$, then the bound $\lfloor n/2 \rfloor$ is appropriate; and for $r > n/2$, the bound $(n-r)$ is appropriate. The last bound has an important consequence. If the number of revoked users is greater than $n/2$, it may appear that using individual transmission to the privileged users would be better than using the CTSD method. But, The bound of $(n-r)$ on the header size shows that this is not true. Using the CTSD method is never worse than individual transmission to privileged users.

The bound of Theorem 3 holds for the SD scheme, i.e., for full trees. The only previously proved upper bound for the SD scheme is $2r-1$. The other two bounds do not appear to have been reported with proofs in the literature. In fact, there does not seem to be an easy way to argue about these bounds without using the recurrences that we have derived.

Fix a value for r and denote by n_r the minimum value of n such that there exists an (n,r)-revocation pattern giving rise to a header of size $2r-1$. Theorem 4 characterizes n_r.

Theorem 4. *In the CTSD method, let $2^{k-1} < r \le 2^k$. When $r \le 2^{k-1} + 2^{k-2}$, let $r_1 = 2^{k-2}$ and $r_0 = r - 2^{k-2}$ and hence,*

$$n_r = n_{r_0} + 2^{2k-2} + 2^{2k-1}$$

and when $r > 2^{k-1} + 2^{k-2}$, let $r_0 = 2^{k-1}$ and $r_1 = r - 2^{k-1}$ and hence,

$$n_r = 2^{2k-1} + n_{r_1} + 2^{2k-1}.$$

From this it easily follows that for the SD method, for any r in the range $2^{k-1} < r \le 2^k$, $n_r = 2^{2k+1}$. This has been earlier proved in [MMW09].

For the SD scheme the number of users is a power of 2. In this case, we show that the recurrences lead to a generating function for the sequence $N(n,r,h)$. Let the number of users be $n = 2^{\ell_0}$ and hence the tree $\mathcal{T}^0$ is full and of height ℓ_0. For a full tree $\mathcal{T}^0$, all subtrees $\mathcal{T}^i$ are full and at level ℓ, there are $2^{\ell_0-\ell}$ subtrees with 2^ℓ leaves in each.

We define $T_\ell(r,h) = T(2^\ell, r, h)$ and $N_\ell(r,h) = N(2^\ell, r, h)$. Then the recurrences (1) and (2) for counting the number of revocation patterns become.

$$N_{\ell_0}(r,h) = T_{\ell_0}(r,h) + \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} \times T_\ell(r,h-1)\right). \tag{6}$$

$$T_{\ell_0}(r,h) = \sum_{r_1=1}^{r-1} \sum_{h_1=0}^{h} N_{\ell_0-1}(r_1,h_1) \times N_{\ell_0-1}(r-r_1,h-h_1). \tag{7}$$

The following result states the form of the generating function and the proof is given in Appendix B.

Theorem 5. *The generating function for the sequence $N_{\ell_0}(r,h)$ of numbers defined in (6) above, is given by $X_{\ell_0}(x,y)$ where*

$$X_{\ell_0}(x,y) = \left(X_{\ell_0-1}(x,y) - xy^{2^{\ell_0-1}}\right)^2 + xy^{2^{\ell_0}} + 2^{\ell_0}x^2y^{2^{\ell_0}-1} + \sum_{\ell=1}^{\ell_0-1}\left(2^{\ell_0-\ell}xy^{2^{\ell_0}-2^{\ell}} \times \left(X_{\ell-1}(x,y) - xy^{2^{\ell-1}}\right)^2\right).$$

A similar generating function was found by Park and Blake in [PB06]. It was directly derived based on the structural properties of the tree. We have taken a different approach of first finding the recurrence relations for the sequence $N(n,r,h)$ and then deriving the generating function from it.

5 Expected Header Length in the CTSD and SD methods

In the previous section, we have studied upper bounds on the header length. In practice, however, it is of interest to know the average header length. This will provide a broadcast centre with valuable information about the average communication bandwidth.

Given the number of users n such that $2^{\ell_0-1} < n \leq 2^{\ell_0}$, and the number of revoked users r, there are $\binom{n}{r}$ possible revocation patterns. Each such revocation pattern gives rise to a subset cover for the privileged users and hence a header in the ciphertext C. We now obtain an algorithm to compute the expected header length for a given n and r in the CTSD scheme. In particular this algorithm applies to the SD method and is of significant practical interest.

The Random Experiment: *We consider the random experiment where r out of the n initially un-revoked leaves of the tree $\mathcal{T}^0$ are chosen uniformly at random without replacement and revoked.* This gives rise to a random (n,r)-revocation pattern and hence a corresponding random subset cover S_c and its header length h. Let $X_{n,r}$ be the random variable taking the value of the header length h due to the (n,r)-revocation pattern of the above experiment. Next, we associate a random variable with each node of the tree $\mathcal{T}^0$. Let $X^i_{n,r} \in \{0,1\}$ be a random variable associated with node i of $\mathcal{T}^0$. $X^i_{n,r} = 1$ denotes the event that the cover contains a subset $S_{i,j} = \mathcal{T}^i \setminus \mathcal{T}^j$ where j is some node in the subtree $\mathcal{T}^i$. In other words, when $X^i_{n,r} = 1$ we say that node i generates a subset for the cover. Similarly, $X^i_{n,r} = 0$ denotes the event that there is no subset $S_{i,j}$ in the cover. Since i is also represented by (ℓ_i, k_i), $X^i_{n,r}$ will also be written as $X^{\ell_i,k_i}_{n,r}$ whenever the nodes need to be viewed level-wise and is appropriate in the context.

The Expected Header Length: Since the header constitutes of subsets $S_{i,j}$, each rooted at a different node i, it is easy to see that, $X_{n,r} = X^0_{n,r} + X^1_{n,r} + \ldots + X^{n-2}_{n,r}$. By linearity of expectation:

$$E[X_{n,r}] = E[X^0_{n,r}] + E[X^1_{n,r}] + \ldots + E[X^{n-2}_{n,r}]. \tag{8}$$

Since all the random variables $X^t_{n,r}$ follow a *Bernoulli distribution with probability* $\Pr[X^t_{n,r} = 1]$, we get:

$$E[X_{n,r}] = \Pr[X^0_{n,r} = 1] + \Pr[X^1_{n,r} = 1] + \ldots + \Pr[X^{n-2}_{n,r} = 1]. \tag{9}$$

Calculating each of these $n-1$ probability terms individually would give the expected header length. However, the running time can be optimized. Recall that $\mathcal{P}^0$ is the unique path from the root to a leaf node which contains the nodes at which the non-full subtrees of $\mathcal{T}^0$ are rooted. As we had discussed before, the subtrees $\mathcal{T}^i$ for which i is not on $\mathcal{P}^0$ are full. For a level ℓ of $\mathcal{T}^0$ the subtrees to the left of $\mathcal{P}^0$ are all full and have equal number of leaves. Hence, $\Pr[X^i_{n,r} = 1]$ needs to be computed only once for every such node i to the left of $\mathcal{P}^0$ at level ℓ. Similarly for nodes to the right of $\mathcal{P}^0$. Hence, efficient computation of $E[X_{n,r}]$ using (9), boils down to finding

$\Pr[X^j_{n,r} = 1]$ level-wise. There are q_ℓ internal nodes at all levels $\ell \geq 2$. At level 1, there are $n - 2^{\ell_1} = q_0/2$ internal nodes. The other $q_1 - (n - 2^{\ell_1})$ nodes at level 1 are leaves. Hence, (9) can also be written as:

$$E[X_{n,r}] = \sum_{\ell=2}^{\ell_0} \sum_{k=1}^{q_\ell} \Pr[X^{\ell,k}_{n,r} = 1] + \sum_{k=1}^{q_0/2} \Pr[X^{1,k}_{n,r} = 1]. \tag{10}$$

When $r = 0$, there is only one set $\mathcal{N}$ in the cover $\mathcal{S}_c$ and hence, $E[X_{n,0}] = 1$. Here on, we will consider $r \geq 1$.

$\Pr[X^{\ell_i,k_i}_{n,r} = 1]$ for the node i of $\mathcal{T}^i$: The sibling subtree $\mathcal{T}^s$ of node i may be $\mathcal{T}^{i-1}$ on its left or $\mathcal{T}^{i+1}$ on its right. To find the probability that node i generates a subset $S_{i,j}$ for the cover, we observe that the event $X^i_{n,r} = 1$ occurs when the sibling subtree $\mathcal{T}^s$ of i has at least one revoked node and exactly one of the subtrees of i has at least one revoked user. We define the events R^i_{sb}, R^i_{lt} and R^i_{rt} for node i with respect to our random experiment. R^i_{sb} denotes the event that the number of revoked nodes in the sibling subtree of $\mathcal{T}^i$ is non-zero. R^i_{lt} (respectively R^i_{rt}) denotes the event that the number of revoked nodes in the left (respectively right) subtree $\mathcal{T}^{2i+1}$ (respectively $\mathcal{T}^{2i+2}$) is non-zero.

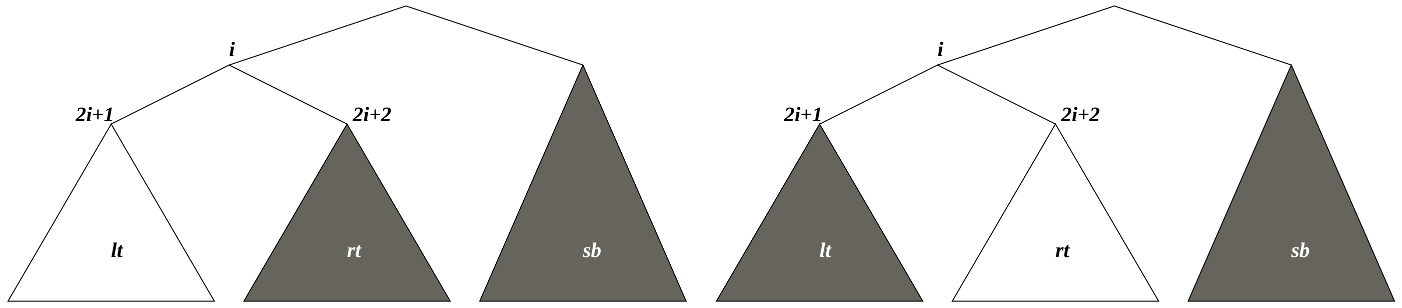

Figure 4: Figures demonstrating the events $R^i_{sb} \wedge R^i_{rt} \wedge \overline{R^i_{lt}}$ and $R^i_{sb} \wedge R^i_{lt} \wedge \overline{R^i_{rt}}$ respectively. The triangles represent subtrees rooted at the respective nodes. White denotes that the subtree has no revoked user in it. Gray denotes that the subtree has at least one revoked user in it. The sizes of the subtrees are not to the scale of the number of users in them.

Lemma 6. *For an internal non-root node i in $\mathcal{T}^0$, the probability that the cover S_c contains a set of the form $\mathcal{T}^i \setminus \mathcal{T}^j$ where j is some node in the subtree $\mathcal{T}^i$, is given by $\Pr[X^i_{n,r} = 1]$ where*

$$\Pr[X^i_{n,r} = 1] = \Pr[R^i_{sb} \wedge R^i_{rt} \wedge \overline{R^i_{lt}}] + \Pr[R^i_{sb} \wedge R^i_{lt} \wedge \overline{R^i_{rt}}].$$

For the root node 0, this probability is given by $\Pr[X^0_{n,r} = 1]$ where

$$\Pr[X^0_{n,r} = 1] = \Pr[\overline{R^0_{lt}}] + \Pr[\overline{R^0_{rt}}].$$

Proof. For a non-root node i, a subset $S_{i,j}$ occurs in the cover when there is at least one revoked user in exactly one of the subtrees $\mathcal{T}^{2i+1}$ or $\mathcal{T}^{2i+2}$ of i. The sibling subtree $\mathcal{T}^s$ should also have at least one revoked user. Hence the event $X^i_{n,r} = 1$ can be divided into two mutually exclusive and exhaustive events. First, when the sibling subtree and the right subtree of $\mathcal{T}^i$ have at least one revoked user in each and the left subtree does not have any: $(R^i_{sb} \wedge R^i_{rt} \wedge \overline{R^i_{lt}})$. Second, when the sibling subtree and the left subtree of $\mathcal{T}^i$ have at least one revoked user in each and the right subtree does not have any: $(R^i_{sb} \wedge R^i_{lt} \wedge \overline{R^i_{rt}})$.

The root node 0 does not have any sibling subtree. Hence the event $X^0_{n,r} = 1$ occurs when all revoked users are either in the left or right subtree of 0. Hence the lemma. □

To simplify the computation of these probabilities in Lemma 6, we define a new notation $\eta_r(\alpha, \beta)$ to indicate *the probability of choosing r elements from a set of α elements such that β out of these α elements are never chosen.* So, if $\beta \geq \alpha - r + 1$, then $\eta_r(\alpha, \beta) = 0$ by definition. Else, for $0 < \beta < \alpha - r + 1$,

$$\eta_r(\alpha, \beta) = \frac{\binom{\alpha-\beta}{r}}{\binom{\alpha}{r}} = \left(1 - \frac{\beta}{\alpha}\right)\left(1 - \frac{\beta}{\alpha - 1}\right)\left(1 - \frac{\beta}{\alpha - 2}\right)\cdots\left(1 - \frac{\beta}{\alpha - r + 1}\right). \tag{11}$$

Theorem 7. *For an internal non-root node i of $\mathcal{T}^0$ whose sibling subtree has λ_s leaves,*

$$\begin{aligned}\Pr[X^i_{n,r} = 1] = {} & \eta_r(n, \lambda_{2i+1}) + \eta_r(n, \lambda_{2i+2}) - \eta_r(n, \lambda_s + \lambda_{2i+1}) - \eta_r(n, \lambda_s + \lambda_{2i+2}) \\ & - 2\eta_r(n, \lambda_{2i+1} + \lambda_{2i+2}) + 2\eta_r(n, \lambda_s + \lambda_{2i+1} + \lambda_{2i+2}).\end{aligned} \tag{12}$$

For the root node 0 of $\mathcal{T}^0$,

$$\Pr[X^0_{n,r} = 1] = \eta_r(n, \lambda_1) + \eta_r(n, \lambda_2). \tag{13}$$

Proof. The following two expressions can be obtained by usual probability arguments.

$$\left.\begin{array}{rcl}\Pr[R^i_{sb} \wedge R^i_{rt} \wedge \overline{R^i_{lt}}] & = & \Pr[\overline{R^i_{lt}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{lt}}] - \Pr[\overline{R^i_{rt}} \wedge \overline{R^i_{lt}}] + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{rt}} \wedge \overline{R^i_{lt}}]; \\ \Pr[R^i_{sb} \wedge R^i_{lt} \wedge \overline{R^i_{rt}}] & = & \Pr[\overline{R^i_{rt}}] - \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{rt}}] - \Pr[\overline{R^i_{lt}} \wedge \overline{R^i_{rt}}] + \Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{lt}} \wedge \overline{R^i_{rt}}].\end{array}\right\} \tag{14}$$

Next, we deduce the expression for finding $\Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{lt}} \wedge \overline{R^i_{rt}}]$ in terms of $\eta_r(\cdot, \cdot)$. This is the probability of choosing r elements from n such that none of the users in the subtrees $\mathcal{T}^{2i+1}$, $\mathcal{T}^{2i+1}$ or the sibling subtree $\mathcal{T}^s$ of i are chosen. Consequently, $\Pr[\overline{R^i_{sb}} \wedge \overline{R^i_{lt}} \wedge \overline{R^i_{rt}}] = \eta_r(n, \lambda_s + \lambda_{2i+1} + \lambda_{2i+2})$. The other probabilities on the right hand sides of (14) can be found similarly by excluding the users in the respective subtrees. From Lemma 6, and substituting the probabilities on the right hand sides of (14) with their corresponding $\eta_r(\cdot, \cdot)$ equivalents, we get:

$$\begin{aligned}\Pr[X^i_{n,r} = 1] = {} & \eta_r(n, \lambda_{2i+1}) + \eta_r(n, \lambda_{2i+2}) - \eta_r(n, \lambda_s + \lambda_{2i+1}) - \eta_r(n, \lambda_s + \lambda_{2i+2}) \\ & - 2\eta_r(n, \lambda_{2i+1} + \lambda_{2i+2}) + 2\eta_r(n, \lambda_s + \lambda_{2i+1} + \lambda_{2i+2}).\end{aligned} \tag{15}$$

For the root node, $\Pr[X^0_{n,r} = 1] = \Pr[\overline{R^0_{lt}}] + \Pr[\overline{R^0_{rt}}]$ where $\Pr[\overline{R^0_{lt}}] = \eta_r(n, \lambda_1)$ and $\Pr[\overline{R^0_{rt}}] = \eta_r(n, \lambda_2)$. Hence,

$$\Pr[X^0_{n,r} = 1] = \eta_r(n, \lambda_1) + \eta_r(n, \lambda_2). \tag{16}$$

□

The algorithm for computing $E[X_{n,r}]$: Now that we have the expressions to find $\Pr[X^i_{n,r} = 1]$ for all $i \in \{0, \ldots, n-2\}$ in Theorem 7, the values for λ_s, λ_{2i+1} and λ_{2i+2} for node i have to substituted appropriately in (15) and (16). By doing these substitutions for nodes at each level $\ell \in \{1, \ldots, \ell_0\}$ of $\mathcal{T}^0$, we get the complete algorithm. For level $\ell \in \{2, \ldots, \ell_0 - 1\}$, this computation is done in four steps: (1) for the node $k^{\mathcal{P}}_\ell$ of level ℓ, (2) its sibling subtree, (3) all full subtrees to the left of the above two subtrees, and (4) all full subtrees to the right of the two subtrees in 1 and 2. The subtree at position $k^{\mathcal{P}}_\ell$ at level ℓ is the only possible non-full subtree for level ℓ and is of height ℓ. If $k^{\mathcal{P}}_\ell$ is odd, its sibling subtree is full and of height $\ell - 1$. If $k^{\mathcal{P}}_\ell$ is even, its sibling subtree is full and of height ℓ. The subtree at node $k^{\mathcal{P}}_{\ell-1}$ of level $\ell - 1$ is always a subtree of the tree rooted at node $k^{\mathcal{P}}_\ell$ of level ℓ. When $k^{\mathcal{P}}_{\ell-1}$ is odd, the right subtree of the tree rooted at node $k^{\mathcal{P}}_\ell$ of level ℓ is full. When $k^{\mathcal{P}}_{\ell-1}$ is even, the left subtree of the tree rooted at node $k^{\mathcal{P}}_\ell$ of level ℓ is full. For the root node 0 and the nodes at level 1, the substitutions are more simple.

To analyze the running time of the algorithm, we observe that each computation of $\eta_r(\alpha, \beta)$ involves $O(r)$ multiplications and there are a constant number of computations of $\eta_r(\alpha, \beta)$ for each level of the tree. Hence, the algorithm requires $O(r \log n)$ multiplications and $O(1)$ space.

Remarks.

Simulation method for estimating the expected header length: Suppose it is desired to obtain an idea of the average header length for n users of which r are revoked. One can choose k random revocation patterns. For each such pattern, the actual header generation algorithm is executed and the header size is obtained. The average header size over the k patterns provides an idea of the average header length. This method, however, is less efficient than our algorithm to compute the expected header length. For each of the k revocation patterns, the simulation will have to construct the Steiner Tree to compute the generated subsets. Each such run will require $\Omega(r^2 \log n)$ memory accesses and $O(n)$ space for finding the cover and hence the header length. In comparison, our algorithm requires $O(r \log n)$ multiplications and $O(1)$ space and finds the exact header length. Further, it is much simpler to implement.

On the other hand, there is a situation where the simulation method may be useful. For the probability analysis, it is usual to assume that revocations take place uniformly. In practice, though, this may not be true. For non-uniform distributions, mathematical analysis may not be possible. For such situations, there is no other option but to use the simulation method to get an idea of the average header length. Additionally, simulations may provide more information about the probability distribution than just the average header length.

Approximation: In [PB06] a formula is given for the expected header length. However, they 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 computes the exact value of the expected header length. Also, [PB06] work only with the SD scheme and so their results do not apply when the number of users is not a power of two.

We have implemented our algorithm to compute the expected header length. Table 3 shows that as r goes above a certain minimum, the expected header length of the CTSD method is significantly better than the SD method. To summarize, the CTSD algorithm always gives better transmission efficiency and its cumulative improvement over many messages is significant on the bandwidth. Since replacing the SD algorithm with the CTSD scheme can be done with very little additional cost the CTSD algorithm should be the more efficient and practical choice.

5.1 Asymptotic Analysis of the Expected Header Length for the SD Method

It is of interest to find the maximum possible value of the expected header length. We carry out this task for full binary trees. In this case, the CTSD method becomes the SD method.

For $n = 2^{\ell_0}$, for any internal node $i \in \{0, \ldots, n-2\}$, $\lambda_{2i+1} = \lambda_{2i+2} = 2^{\ell_i - 1}$. For any node at level $\ell_i > 0$, $\lambda_s = 2^{\ell_i + 1}$. Substituting these values for a node (ℓ, k), (15) becomes:

$$\Pr[X_{n,r}^{\ell,k} = 1] = 2[\eta_r(n, 2^{\ell-1}) - \eta_r(n, 2 \times 2^{\ell-1}) - \eta_r(n, 3 \times 2^{\ell-1}) + \eta_r(n, 4 \times 2^{\ell-1})]. \tag{17}$$

This probability is independent of k. In other words, the probability of generating a subset for the cover is equal for all nodes at level ℓ. Define

$$B_{n,r}^{(\ell)} \triangleq \Pr[X_{n,r}^{\ell,k} = 1] = 2[\eta_r(n, 2^{\ell-1}) - \eta_r(n, 2 \times 2^{\ell-1}) - \eta_r(n, 3 \times 2^{\ell-1}) + \eta_r(n, 4 \times 2^{\ell-1})]. \tag{18}$$

Note that by this definition, for the only node at level ℓ_0, i.e., the root node, $B_{n,r}^{(\ell_0)} = 2\eta_r(n, 2^{\ell_0 - 1})$ which is consistent with (16) for $n = 2^{\ell_0}$. For a given $n = 2^{\ell_0}$ and r, the expected header length $H_{n,r}$ due to the subset cover algorithm of the CTSD scheme is defined as:

$$H_{n,r} \triangleq E[X_{n,r}] = \sum_{\ell=1}^{\ell_0} 2^{\ell_0 - \ell} B_{n,r}^{(\ell)}.$$

r	$n < 2^{\ell_0}$ (CTSD)	CTSD $E[X_{n,r}]$	$n = 2^{\ell_0}$ (SD)	SD $E[X_{n,r}]$	Extra KBytes
10^2	$2^{19}+1$	124.49	2^{20}	124.50	0.001KB
10^2	$2^{19}+2^{18}$	124.49	2^{20}	124.50	0.001KB
10^3	$2^{19}+1$	1242.49	2^{20}	1243.80	0.021KB
10^3	$2^{19}+2^{18}$	1243.36	2^{20}	1243.80	0.007KB
5×10^3	$2^{19}+1$	6159.94	2^{20}	6192.74	0.525KB
5×10^3	$2^{19}+2^{18}$	6181.80	2^{20}	6192.74	0.175KB
10^4	$2^{19}+1$	12188.73	2^{20}	12319.86	2.098KB
10^4	$2^{19}+2^{18}$	12276.12	2^{20}	12319.86	0.700KB
10^5	$2^{19}+1$	98555.30	2^{20}	111451.58	206.340KB
10^5	$2^{19}+2^{18}$	107134.01	2^{20}	111451.58	69.081KB
10^5	$2^{23}+1$	122870.35	2^{24}	123690.49	13.122KB
10^5	$2^{23}+2^{22}$	123417.07	2^{24}	123690.49	4.375KB
10^6	$2^{23}+1$	1082115.11	2^{24}	1163305.89	1299.056KB
10^6	$2^{23}+2^{22}$	1136173.35	2^{24}	1163305.89	434.128KB

Table 3: The expected header lengths for the SD and CTSD schemes for different n and r and the number of extra bytes needed per message of broadcast. Here we assume each session key is 128 bits long. The additional number of bytes required by the SD scheme is computed as 16 times the difference in header length of the two schemes.

As n increases, the value of $H_{n,r}$ converges to a limit which depends only on r and is independent of n. The following result gives this limiting upper bound on $H_{n,r}$. We use the notation $x \uparrow a$ to denote that a variable x increases to the limit a.

Theorem 8. *For all $n \geq 1$, $r \geq 1$, the expected header length $H_{n,r} \uparrow H_r$, as n increases through powers of two, where*

$$H_r = 3r - 2 - 3 \times \sum_{i=1}^{r-1} \left(\left(-\frac{1}{2} \right)^i + \sum_{k=1}^{i} (-1)^k \binom{i}{k} \frac{(2^k - 3^k)}{(2^k - 1)} \right).$$

The proof requires a bit of detailed combinatorial analysis and is provided in Appendix C. We have computed the ratio H_r/r for many values of r and have found it to be always less than 1.25.

In [NNL01], a sketchy argument was given to show that $H_{n,r}$ is bounded above by $1.38r$. It was mentioned that simulation results showed a tighter upper bound of $1.25r$. Values computed using Theorem 8 explain this observation. On the other hand, Theorem 8 shows that the actual limiting value for the expected header length is much more complicated than the simple $1.25r$ that was suggested in [NNL01]. Our experiments have shown that the convergence to this limiting value is quite fast. Further, the bound given by Theorem 8 can be computed in $O(r)$ time and $O(1)$ space.

6 Conclusion

We have proposed a new BE scheme which extends the tree-based SD scheme of [NNL01]. The new Complete Tree Subset Difference method is capable of accommodating any arbitrary number of users that may not be a power of two and hence subsumes the SD scheme of [NNL01]. Almost all results of the CTSD scheme that we subsequently prove are also new for the SD scheme.

Detailed combinatorial analysis of the CTSD scheme is done by finding two recurrences to count the number of ways r out of n users can be revoked to result in a subset cover size of h in the CTSD method. Using these recurrences, it is proved that the maximum possible header length for a given r is $2r-1$. This is no worse than the SD scheme even though arbitrary number of users are accommodated. The maximum header length for all r is $\lfloor \frac{n}{2} \rfloor$. The recurrences are the most efficient tool as per our knowledge to generate exhaustive data for the above count. Using the recurrences, we also find and prove the expression for the minimum number of users required to be in a system so that for a given r, the maximum cover size would reach $2r-1$. For n a power of two, a generating function is found for generating the same sequence as the recurrences.

Probabilistic analysis of the revocation patterns in the CTSD scheme gives the most important result of this work: an efficient algorithm to compute the expected header length for a given n and r. Using this algorithm, it is shown that for practical values of n and r, the CTSD scheme provides better transmission efficiency as compared to the SD scheme. An asymptotic analysis is done using this algorithm that not only gives theoretical support to the empirical upper bound of $1.25r$ mentioned in [NNL01], but also gives an expression to compute the maximum possible expected header length for a given r in the SD algorithm in $O(r)$ time.

Acknowledgement: We thank the anonymous reviewers for their comments that has helped to improve the paper.

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.

[Asa02] Tomoyuki Asano. A revocation scheme with minimal storage at receivers. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2002.

[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.

[BS11] Sanjay Bhattacherjee and Palash Sarkar. An analysis of the Naor-Naor-Lotspiech Subset Difference algorithm (for possibly incomplete binary trees). In Daniel Augot and Anne Canteaut, editors, *Workshop on Coding and Cryptography, April 11-15, 2011*, Workshop on Coding and Cryptography, pages 483–492. INRIA, 2011.

[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.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

[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.

[JG04] Shaoquan Jiang and Guang Gong. Multi-service oriented broadcast encryption. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP*, volume 3108 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2004.

[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.

[LT08] Yi-Ru Liu and Wen-Guey Tzeng. Public key broadcast encryption with low number of keys and constant decryption time. In Ronald Cramer, editor, *Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 380–396. Springer, 2008.

[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.

[PPS11] Duong Hieu Phan, David Pointcheval, and Mario Strefler. Security notions for broadcast encryption. In Javier Lopez and Gene Tsudik, editors, *ACNS*, volume 6715 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2011.

[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 Upper Bounds on Header Length of the SD and CTSD methods

The result below shows that the header length of the CTSD scheme is upper bounded by $2r-1$.

Lemma 9. *$N(\lambda_i, r_1, h_1) = 0$ when $h_1 > 2r_1 - 1$. $T(\lambda_i, r_1, h_1) = 0$ when $h_1 \geq 2r_1 - 1$.*

Proof. First we show that $T(\lambda_i, r_1, h_1) = 0$ when $h_1 \geq 2r_1 - 1$ in (1). We prove this from (2) by induction on r_1. The boundary conditions have been listed in Table 2. We know that, $2^{\ell_i - 1} < \lambda_i \leq 2^{\ell_i}$. By induction hypothesis, when $h' > 2r' - 1$ and $1 \leq r' < r_1$, $N(\lambda_{2i+1}, r', h') = 0$. If $h' \leq 2r' - 1$, then $h_1 - h' > 2r_1 - 1 - h' \geq 2r_1 - 1 - 2r' + 1 = 2(r_1 - r')$. Then, again by induction hypothesis, $N(\lambda_{2i+2}, r_1 - r', h_1 - h') = 0$. Hence, when $h_1 \geq 2r_1 - 1$, $T(\lambda_i, r_1, h_1) = 0$.

Now, if $h_1 > 2r_1 - 1$, the other terms on the right hand side of (1) are $T(\lambda_i, r_1, h_1 - 1)$ where $h_1 - 1 \geq 2r_1 - 1$ for all terms and hence are all 0 as proved above. Hence, when $h_1 > 2r_1 - 1$, $N(\lambda_i, r_1, h_1) = 0$. □

We later show that for sufficiently large n, $N(n, r, 2r-1)$ is positive and also characterize the minimum n for which this happens. Next, we show that $N(n, r, h)$ is monotonic on n for fixed r and h.

Lemma 10. *Let $n_1 \geq n_2$. If $N(n_2, r, h) \neq 0$ then $N(n_1, r, h) \neq 0$. If $T(n_2, r, h) \neq 0$ then $T(n_1, r, h) \neq 0$.*

Proof. Let $T(n_2, r, h) \neq 0$. From (2) we get:

$$T(n_2, r, h) = \sum_{r'=1}^{r-1} \sum_{h'=0}^{h} N(\lambda_1, r', h') \times N(\lambda_2, r - r', h - h').$$

Let $RH = \{(r_1, h_1) \ldots, (r_s, h_s)\}$ be such that both $N(\lambda_1, r', h')$ and $N(\lambda_2, r - r', h - h')$ are non-zero (and hence $N(\lambda_1, r', h') \times N(\lambda_2, r - r', h - h')$ is non-zero) when $(r', h') \in RH$. Hence, we can also write:

$$T(n_2, r, h) = \sum_{(r',h') \in RH} N(\lambda_1, r', h') \times N(\lambda_2, r - r', h - h').$$

Since $\lambda_1 < n_2$ (by the structure of $\mathcal{T}^0$ with n_2 leaves), hence by induction hypothesis, for any $\lambda \geq \lambda_1$, $N(\lambda_1, r, h) \neq 0$ implies $N(\lambda, r, h) \neq 0$. Similarly, since $\lambda_2 < n_2$, hence by induction hypothesis, for any $\lambda \geq \lambda_2$, $N(\lambda_2, r, h) \neq 0$ implies $N(\lambda, r, h) \neq 0$. When there are n_1 leaves in the tree let there be λ'_1 leaves in the left subtree and λ'_2 leaves in the right subtree of the root node. Hence, by the construction of $\mathcal{T}^0$, we get $\lambda'_1 \geq \lambda_1$ and $\lambda'_2 \geq \lambda_2$. In the expression for $T(n_1, r, h)$, for $(r', h') \in RH$, by induction hypothesis, $N(\lambda'_1, r', h')$ and $N(\lambda'_2, r - r', h - h')$ are both non-zero. Hence, for at least $(r', h') \in RH$, $N(\lambda'_1, r', h') \times N(\lambda'_2, r - r', h - h')$ is non-zero. Thus, $T(n_1, r, h) \neq 0$.

Now, let $N(n_2, r, h) \neq 0$. From (1) we get:

$$N(n_2, r, h) = T(n_2, r, h) + \sum_{j=1}^{n_2-2} T(\lambda_j, r, h - 1).$$

Let $I = \{i_1, \ldots, i_t\}$ be the nodes of $\mathcal{T}^0$ (with n_2 leaves) such that $T(\lambda_i, r, h) \neq 0$ for $i \in I$. By induction hypothesis, for any $\lambda_j < n_2$ and $\lambda_i > \lambda_j$, if $T(\lambda_j, r, h) \neq 0$ then $T(\lambda_i, r, h) \neq 0$. Hence, we can also write:

$$N(n_2, r, h) = T(n_2, r, h) + \sum_{i \in I} T(\lambda_i, r, h - 1).$$

Here, $T(n_2, r, h) \neq 0$ implies $T(n_1, r, h) \neq 0$ by the first part of this proof. By the construction of the tree $\mathcal{T}^0$, $\lambda'_i \geq \lambda_i$ where λ'_i is the number of leaves in the subtree rooted at node i of the tree $\mathcal{T}^0$ for n_1 leaves. By induction hypothesis, at least for $i \in I$, since $T(\lambda_i, r, h - 1) \neq 0$, hence $T(\lambda'_i, r, h - 1) \neq 0$. Thus, $N(n_1, r, h) \neq 0$.

□

Now, we prove that if r is not small compared to n, then $T(n, r, 2r - 2) = 0$.

Lemma 11. *For $n \leq 2^{2k+1}$ and $r > 2^k$, $T(n, r, 2r - 2) = 0$.*

Proof. For $T(n, r, 2r - 2)$ in (2), let $h' < 2r' - 1$, then $h - h' = 2r - 2 - h' > 2r - 2 - 2r' + 1 = 2(r - r') - 1$. Hence by Lemma 9, $N(\lambda_2, r - r', h - h') = 0$. Similarly, if $h' > 2r' - 1$, $N(\lambda_1, r', h') = 0$. So, in the expression for $T(n, r, 2r - 2)$, the terms on the right hand side of (2) are 0 if $h' \neq 2r' - 1$. Hence,

$$T(n, r, 2r - 2) = \sum_{r'=1}^{r-1} N(\lambda_1, r', 2r' - 1) \times N(\lambda_2, r - r', 2(r - r') - 1). \tag{19}$$

Now by induction on λ_i, we prove that $N(\lambda_1, r', 2r' - 1) = 0$ and $N(\lambda_2, r - r', 2(r - r') - 1) = 0$. The boundary conditions have been listed in Table 2. By induction hypothesis, for $\lambda_i \leq 2^{2m+1}$ where $m < k$ and $r' > 2^m$ let us assume $T(\lambda_i, r', 2r' - 2) = 0$. In (19), let $r' \geq \frac{r}{2}$ which implies $r' > 2^{k-1}$. Hence, for $\lambda_i \leq 2^{2k-1}$, $T(\lambda_i, r', 2r' - 2) = 0$ by the induction hypothesis. Also, by Lemma 9, $T(\lambda_1, r', 2r' - 1) = 0$. Putting these values in (1), we get $N(\lambda_1, r', 2r' - 1) = 0$. Similarly, for $r - r' \geq \frac{r}{2}$ which implies $r - r' > 2^{k-1}$, we get $N(\lambda_2, r - r', 2(r - r') - 1) = 0$. Hence, from (19) $T(n, r, 2r - 2) = 0$. □

Some insight: Given a revocation pattern, if we revoke one more user from it, that can result in either increase, decrease or no change in the cover size. An increase in cover size mostly happens when the newly revoked user is not adjacent to any previously revoked user. The cover size remains unchanged or decreases when the newly revoked user is adjacent to a previously revoked user. Decrease in cover size happens when the user in a singleton subset of the cover is revoked. As the number of revoked users increase, the maximum possible cover size for that number of revoked users increases up to a certain point. After that the maximum possible cover size decreases. One may also observe that for $n > 2$, i.e., $\ell_1 \geq 1$, $q_0/2 = n - 2^{\ell_1}$. Since 2^{ℓ_1} is even for $\ell_1 \geq 1$, hence when n is even $q_0/2$ is even and when n is odd $q_0/2$ is odd.

Lemma 12. *The header length in the CTSD method for n users is at most $\lfloor \frac{n}{2} \rfloor$ irrespective of the number of revoked users.*

Proof. First, we show that $N(n, r, h) = 0$ for $h > \frac{n}{2}$ for any r. We prove this by induction on n. From (1) we have:

$$N(n, r, h) = T(n, r, h) + \sum_{i=1}^{n-2} T(\lambda_i, r, h-1)$$

and hence, $T(n, r, h) \leq N(n, r, h)$. When $\lambda_i < n$ and $h-1 \geq \lfloor \frac{n}{2} \rfloor$, $N(\lambda_i, r, h-1) = 0$. Thus, $\sum_{i=1}^{n-2} T(\lambda_i, r, h-1) = 0$. From (2) we get:

$$T(n, r, h) = \sum_{r'=1}^{r-1} \sum_{h'=0}^{h} N(\lambda_1, r', h') \times N(\lambda_2, r - r', h - h').$$

When $h' > \frac{\lambda_1}{2}$, $N(\lambda_1, r', h') = 0$ by induction hypothesis. When $h' \leq \frac{\lambda_1}{2}$, since $h > \frac{n}{2}$, $h - h' > \frac{n}{2} - \frac{\lambda_1}{2} = \frac{\lambda_2}{2}$. Therefore, $N(\lambda_2, r - r', h - h') = 0$ by induction hypothesis. Hence, $N(n, r, h) = 0$ for $h > \frac{n}{2}$ for any r.

Next, we show that the upper bound of $\lfloor \frac{n}{2} \rfloor$ is actually achieved. First let us assume that n is even and hence $q_0/2$ is even. We construct a revocation pattern such that none of the users are revoked initially. Now, let us form a revocation pattern by revoking one user from each of the $q_0/2$ subtrees rooted at level q_1 with leaves at level q_0 and one user each from subtrees rooted at level 2 with leaves at level 1. Since all the privileged users would form singleton subsets in the cover for this revocation pattern, hence the header length for the revocation pattern thus constructed is of size q_1 $(= \frac{n}{2})$. Now, if we attempt to revoke any other user, then by pigeonhole principle, one of the sets in the cover gets removed and hence the header length decreases. Hence, for even n, the maximum header length is $\frac{n}{2}$.

For odd n, $q_0/2$ is odd. We construct a revocation pattern similarly by revoking one user from each of the $q_0/2$ subtrees rooted at level q_1 with leaves at level q_0 and one user each from subtrees rooted at level 2 with leaves at level 1. Since $q_0/2$ is odd, there will be one subtree with leaves at both levels 0 and 1. This subtree is rooted at the node at position $k_2^{\mathcal{P}}$. For this subtree, only one out of the three users in it is revoked. All the privileged users other than the one generated from the above subtree would form singleton subsets. Hence the cover size for the revocation pattern thus constructed is of size q_1 $(= \lfloor \frac{n}{2} \rfloor)$. This is again the maximum header length by the same argument as above.

Hence, the maximum header length is $\lfloor \frac{n}{2} \rfloor$ for n users. □

The complete upper bound on the header length is given in Theorem 3 stated in Section 4.4.

Proof of Theorem 3.

Proof. The bounds $2r - 1$ and $\lfloor n/2 \rfloor$ have already been shown. We show the bound of $n - r$ on the header size. The proof of this is similar to the first part of the proof of Lemma 12, i.e., we show that $N(n, r, h) = 0$ for $h > n - r$.

For $\lambda_i < n$, we have $h - 1 > n - 1 - r \geq \lambda_i - r$ and hence using induction, $N(\lambda_i, r, h-1) = 0$ which implies that $T(\lambda_i, r, h-1)$ is also zero. Again, consider the value of $T(n, r, h)$ and the recurrence expressing this in terms of $N(\lambda_1, r', h')$ and $N(\lambda_2, r-r', h-h')$, where $\lambda_1+\lambda_2 = n$. If $h' > \lambda_1 - r'$, then using induction, $N(\lambda_1, r', h') = 0$. So, suppose that $h' \leq \lambda_1 - r'$. Using $h > n - r$, we have $h - h' > (n - \lambda_1) - (r - r') = \lambda_2 - (r - r')$ and again using induction, $N(\lambda_2, r - r', h - h') = 0$.

This shows that $T(n, r, h) = 0$ which combined with the fact that the other relevant values of $T(\cdot, \cdot, \cdot)$ are zero, shows that $N(n, r, h) = 0$ for $h > n - r$. □

Recall that n_r is the minimum number of users so that there exists an (n, r)-revocation pattern which is covered by a header of length $2r - 1$. Lemma 9 shows that the upper bound on the header length is $2r - 1$. By characterizing n_r we show that this upper bound on h is actually achieved.

Lemma 13. *In the CTSD method, $2^{k-1} < r \leq 2^k$ if and only if $2^{2k} < n_r \leq 2^{2k+1}$.*

Proof. We first prove that if $2^{k-1} < r \leq 2^k$, then $2^{2k} < n_r \leq 2^{2k+1}$ (by showing that $N(2^{2k}, r, 2r-1) = 0$ and $N(2^{2k+1}, r, 2r-1) \neq 0$). Although by Lemma 9, $T(2^{2k+1}, r, 2r-1) = 0$, we show that $T(2^{2k}, r, 2r-2) \neq 0$ and hence at least one of the terms on the right hand side of (1) is non-zero and hence $N(2^{2k+1}, r, 2r-1) \neq 0$. From (2) we get:

$$T(2^{2k}, r, 2r-2) = \sum_{r'=1}^{r-1} \sum_{h'=0}^{2r-2} N(2^{2k-1}, r', h') \times N(2^{2k-1}, r-r', 2r-2-h').$$

When $h' > 2r'-1$, $N(2^{2k-1}, r', h') = 0$ by Lemma 9. Similarly, when $h' < 2r'-1$, $2r-2-h' > 2r-2-2r'+1 = 2(r-r')-1$ and hence $N(2^{2k-1}, r-r', 2r-2-h') = 0$. Hence, we get

$$T(2^{2k}, r, 2r-2) = \sum_{r'=1}^{r-1} N(2^{2k-1}, r', 2r'-1) \times N(2^{2k-1}, r-r', 2(r-r')-1).$$

When $r' = \lceil \frac{r}{2} \rceil$ $(2^{k-2} < r' \leq 2^{k-1})$ by induction hypothesis, $n_{r'} \leq 2^{2k-1}$ and hence by Lemma 10, both $N(2^{2k-1}, r', 2r'-1)$ and $N(2^{2k-1}, r-r', 2(r-r')-1)$ are non-zero. Hence, $T(2^{2k}, r, 2r-2) \neq 0$ which implies $N(2^{2k+1}, r, 2r-1) \neq 0$. Since $T(n_r, r, 2r-1) = 0$ and $T(2^{2k-1}, r, 2r-2) = 0$ hence, $n_r < 2^{2k+1}$. Next, we show that $N(2^{2k}, r, 2r-1) = 0$. By Lemma 9, $T(2^{2k}, r, 2r-1) = 0$. By Lemma 11, for all $\lambda_i \leq 2^{2k-1}$ and $r > 2^{k-1}$, $T(\lambda_i, r, 2r-2) = 0$ and hence $N(2^{2k}, r, 2r-1) = 0$.

Next, we prove that for some $2^{2k} < n_r \leq 2^{2k+1}$, the corresponding r is such that $2^{k-1} < r \leq 2^k$. Let the corresponding r be such that $2^{k'-1} < r \leq 2^{k'}$ where $k \neq k'$. Then by the argument above, we know that $2^{2k'} < n_r \leq 2^{2k'+1}$ which is a contradiction since n_r is unique for a given r by definition. Hence the corresponding r is such that $2^{k-1} < r \leq 2^k$. □

Proof of Theorem 4:

Proof. From Lemma 13 we know that for $2^{k-1} < r \leq 2^k$, $2^{2k} < n_r \leq 2^{2k+1}$. For such an n_r, $\lambda_1 = n_r - 2^{2k-1}$ and $\lambda_2 = 2^{2k-1}$. From (1) we get

$$N(n_r, r, 2r-1) = T(n_r, r, 2r-1) + T(n_r - 2^{2k-1}, r, 2r-2) + T(2^{2k-1}, r, 2r-2) + \sum_{i=3}^{n_r-2} T(\lambda_i, r, 2r-2).$$

From Lemma 9 we know that $T(n_r, r, 2r-1) = 0$. From Lemma 11 we know that when $r > 2^{k-1}$ and $\lambda_i \leq 2^{2k-1}$, $T(\lambda_i, r, 2r-2) = 0$. Hence the only non-zero component is $T(n_r - 2^{2k-1}, r, 2r-2)$. From (2) we get

$$N(n_r, r, 2r-1) = T(n_r - 2^{2k-1}, r, 2r-2) = \sum_{r'=1}^{r-1} \sum_{h'=0}^{2r-2} N(\lambda_3, r', h') \times N(\lambda_4, r-r', 2r-2-h').$$

By an argument similar to the one used in the proof for Lemma 13, we get

$$N(n_r, r, 2r-1) = T(n_r - 2^{2k-1}, r, 2r-2) = \sum_{r'=1}^{r-1} N(\lambda_3, r', 2r'-1) \times N(\lambda_4, r-r', 2(r-r')-1).$$

By the construction of $\mathcal{T}^0$ and the fact that $\mathcal{T}^2$ does not have any revoked user, i.e. $T(2^{2k-1}, r, 2r-2) = 0$, it can be seen that $2^{2k-2} < \lambda_3 \leq 2^{2k-1}$ and $2^{2k-2} \leq \lambda_4 < 2^{2k-1}$.

When $r \leq 2^{k-1} + 2^{k-2}$, let $r' = r_0 = r - 2^{k-2}$ and $r - r' = r_1 = 2^{k-2}$. From the construction of the complete tree $\mathcal{T}^0$ for $(n_{r_0} + 2^{2k-2} + 2^{2k-1})$ users, it can be seen that $\lambda_3 = n_{r_0}$ and $\lambda_4 = 2^{2k-2}$. Hence, $N(\lambda_3, r', 2r'-1) = N(n_{r_0}, r_0, 2r_0 - 1) \neq 0$ by the definition of n_r. Also, from Lemma 10 and Lemma 13 we know that for $r = 2^k$ (consequently $n_r < 2^{2k+1}$) and $\lambda \geq 2^{2k+1}$, $N(\lambda, r, 2r-1) \neq 0$. So for $r_1 = r - r' = 2^{k-2}$ and $\lambda_4 = 2^{2(k-2)+2}$ we get, $N(\lambda_4, r-r', 2(r-r')-1) = N(2^{2k-2}, r_1, 2r_1 - 1) \neq 0$. Hence, for $r \leq 2^{k-1} + 2^{k-2}$, $N(n_r, r, 2r-1) \neq 0$ where $n_r = n_{r_0} + 2^{2k-2} + 2^{2k-1}$.

Now, we show that for $2^{k-1} < r \leq 2^{k-1} + 2^{k-2}$ ($r_0 = r - 2^{k-2}$ and $r_1 = 2^{k-2}$), $N(n_r - 1, r, 2r-1) = 0$. In the tree $\mathcal{T}^0$ for $(n_{r_0} + 2^{2k-2} + 2^{2k-1}) - 1$ users, $\lambda_3 = n_{r_0} - 1$ and $\lambda_4 = 2^{2k-2}$. Since there are $n_{r_0} - 1$ users in $\mathcal{T}^3$, at most $r_0 - 1$ revoked users can be accommodated in $\mathcal{T}^3$ so that $N(\lambda_3, r', 2r'-1) \neq 0$ and hence $r' = r_0 - 1$ and $r - r' = 2^{k-2} + 1$. By Lemma 13 for $r - r' > 2^{k-2}$, $n_{r-r'} > 2^{2k-2}$. But, $\lambda_4 = 2^{2k-2}$ and hence $N(\lambda_4, r-r', 2(r-r')-1) = 0$. Consequently, we get $N(n_r - 1, r, 2r-1) = 0$.

When $r > 2^{k-1} + 2^{k-2}$, let $r' = r_0 = 2^{k-1}$ and $r - r' = r_1 = r - 2^{k-1}$. From the construction of the complete tree $\mathcal{T}^0$ for $(2^{2k-1} + n_{r_1} + 2^{2k-1})$ users, it can be seen that $\lambda_3 = 2^{2k-1}$ and $\lambda_4 = n_{r_1}$. Hence, $N(\lambda_4, r-r', 2(r-r')-1) = N(n_{r_1}, r_1, 2r_1 - 1) \neq 0$ by the definition of n_r. From Lemma 10 and Lemma 13 we know that for $r = 2^k$ (consequently $n_r < 2^{2k+1}$) and $\lambda \geq 2^{2k+1}$, $N(\lambda, r, 2r-1) \neq 0$. So for $r_0 = r' = 2^{k-1}$ and $\lambda_3 = 2^{2(k-1)+1}$ we get, $N(\lambda_3, r', 2r'-1) = N(2^{2k-1}, r_0, 2r_0 - 1) \neq 0$. Hence, for $r > 2^{k-1} + 2^{k-2}$, $N(n_r, r, 2r-1) \neq 0$ where $n_r = 2^{2k-1} + n_{r_1} + 2^{2k-1}$.

Now, we show that for $r > 2^{k-1} + 2^{k-2}$, i.e., $r_0 = 2^{k-1}$ and $r_1 = r - 2^{k-1}$, $N(n_r - 1, r, 2r-1) = 0$. In the tree $\mathcal{T}^0$ for $(2^{2k-1} + n_{r_1} + 2^{2k-1}) - 1$ users, $\lambda_3 = 2^{2k-1}$ and $\lambda_4 = n_{r_1} - 1$. Since there are $n_{r_1} - 1$ users in $\mathcal{T}^4$, at most $r_1 - 1$ revoked users can be accommodated in $\mathcal{T}^4$ so that $N(\lambda_4, r-r', 2(r-r')-1) \neq 0$ and hence $r - r' = r_1 - 1$ and $r' = 2^{k-1} + 1$. By Lemma 13 for $r' > 2^{k-1}$, $n_{r'} > 2^{2k-1}$. But, $\lambda_3 = 2^{2k-1}$ and hence $N(\lambda_3, r', 2r'-1) = 0$. Consequently, we get $N(n_r - 1, r, 2r-1) = 0$.

□

B Proof of Theorem 5

Let $X_{\ell_0}(x, y)$ (respectively $Y_{\ell_0}(x, y)$) be the generating function for the sequence $N_{\ell_0}(2^{\ell_0} - r, h)$ (respectively $T_{\ell_0}(2^{\ell_0} - r, h)$).

$$X_{\ell_0}(x, y) = \sum_{r=0}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} N_{\ell_0}(r, h) x^h y^{2^{\ell_0}-r} \qquad Y_{\ell_0}(x, y) = \sum_{r=0}^{2^{\ell_0}} \sum_{h=0}^{2^{\ell_0}-r} T_{\ell_0}(r, h) x^h y^{2^{\ell_0}-r} \tag{20}$$

By definition, when $\ell_0 = 0$, $Y_0(x, y) = 0$ and $X_0(x, y) = 1 + xy$ and when $\ell_0 = 1$, $Y_1(x, y) = 1$ and $X_1(x, y) = 1 + 2xy + xy^2$. Obtaining relations between $X_{l_0}(x, y)$ and $Y_{l_0}(x, y)$ requires long computations. We omit the details of these and report only the relation.

$$Y_{\ell_0}(x, y) = X^2_{\ell_0-1}(x, y) - 2xy^{2^{\ell_0-1}} X_{\ell_0-1}(x, y) - x^2 y^{2^{\ell_0}} = \left(X_{\ell_0-1}(x, y) - xy^{2^{\ell_0-1}}\right)^2. \tag{21}$$

Multiplying both sides of (6) with $x^h y^{2^{\ell_0}-r}$ and summing both sides over $2 \le r \le 2^{\ell_0}$ and $0 \le h \le 2^{\ell_0}$ we get:

$$\sum_{r=2}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} N_{\ell_0}(r,h) x^h y^{2^{\ell_0}-r} = \sum_{r=2}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} T_{\ell_0}(r,h) x^h y^{2^{\ell_0}-r} + \sum_{r=2}^{2^{\ell_0}} \sum_{h=1}^{2^{\ell_0}-r} \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} x^h y^{2^{\ell_0}-r} \times T_\ell(r,h-1)\right). \tag{22}$$

Using a few steps of simplification yields the following relation.

$$X_{\ell_0}(x,y) = xy^{2^{\ell_0}} + 2^{\ell_0} x^2 y^{2^{\ell_0}-1} + Y_{\ell_0}(x,y) + \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} xy^{2^{\ell_0}-2^\ell} \times Y_\ell(x,y)\right). \tag{23}$$

From (21) and (23) above, we get:

$$X_{\ell_0}(x,y) = \left(X_{\ell_0-1}(x,y) - xy^{2^{\ell_0-1}}\right)^2 + xy^{2^{\ell_0}} + 2^{\ell_0} x^2 y^{2^{\ell_0}-1} + \sum_{\ell=1}^{\ell_0-1} \left(2^{\ell_0-\ell} xy^{2^{\ell_0}-2^\ell} \times \left(X_{\ell-1}(x,y) - xy^{2^{\ell-1}}\right)^2\right). \tag{24}$$

C Proof of Theorem 8

Define $D_{n,r} = H_{n,r} - H_{n,r-1}$ and so $H_{n,r} = 1 + \sum_{i=2}^r D_{n,i}$. Using the definition of $B_{n,r}^{(\ell)}$ given by (18) in terms of the η's we get the following expression.

$$D_{n,r+1} = \frac{n}{n-r}\left[-\eta_r(n,1) + \eta_r(n,2) + 3\eta_r(n,3) - 3\sum_{\ell=1}^{\ell_0-1}\left(\eta_r(n, 2\times 2^\ell) - \eta_r(n, 3\times 2^\ell)\right)\right]. \tag{25}$$

Note that for a fixed r, as $n \uparrow \infty$, $\frac{n}{n-r} \uparrow 1$, $\eta_r(n,3) = \frac{(n-3)_r}{(n)_r} \uparrow 1$ and $\eta_r(n,2) - \eta_r(n,1) \uparrow 0$. Hence, as $n \uparrow \infty$, $(-\eta_r(n,1) + \eta_r(n,2) + 3\eta_r(n,3)) \uparrow 3$.

$$\begin{aligned}
\sum_{\ell=1}^{\ell_0-2}\left(\eta_r(n,2\times 2^\ell) - \eta_r(n,3\times 2^\ell)\right) &= \sum_{\ell=1}^{\ell_0-2}\left(\frac{(n-2\times 2^\ell)_r}{(n)_r} - \frac{(n-3\times 2^\ell)_r}{(n)_r}\right) \\
&\ge \frac{1}{(n)_r}\sum_{\ell=1}^{\ell_0-2}\left((n-r+1-2\times 2^\ell)^r - (n-3\times 2^\ell)^r\right) \\
&= \frac{1}{(n)_r}\sum_{\ell=2}^{\ell_0-1}\left(\left(\left(\frac{2^\ell-2}{2^\ell}\right)n - r + 1\right)^r - \left(\left(\frac{2^\ell-3}{2^\ell}\right)n\right)^r\right) \\
&\ge \frac{1}{n^r}\sum_{\ell=2}^{\ell_0-1}\left(\left(\left(\frac{2^\ell-2}{2^\ell}\right)n - r + 1\right)^r - \left(\left(\frac{2^\ell-3}{2^\ell}\right)n\right)^r\right) \\
&= \sum_{\ell=2}^{\ell_0-1}\left(\left(\frac{2^\ell-2}{2^\ell} - \frac{r-1}{n}\right)^r - \left(\frac{2^\ell-3}{2^\ell}\right)^r\right).
\end{aligned} \tag{26}$$

Define K_r as follows:

$$K_r = \lim_{n\to\infty}\sum_{\ell\ge 2}\left(\left(\frac{2^\ell-2}{2^\ell} - \frac{r-1}{n}\right)^r - \left(\frac{2^\ell-3}{2^\ell}\right)^r\right)$$

After several stages of simplifications, the value of K_r is obtained to be the following:

$$K_r = \Big(-\frac{1}{2}\Big)^r + \sum_{k=1}^{r}(-1)^k \binom{r}{k}\frac{(2^k - 3^k)}{(2^k - 1)}. \tag{27}$$

Let $H_r = \lim_{n\to\infty} H_{n,r}$ and $D_r = \lim_{n\to\infty} D_{n,r}$. The proof of Theorem 8 can now be obtained as follows. We have $H_r = 1 + \sum_{i=2}^{r} D_i$. From (25), (26) and (27), we get $D_{r+1} = 3 - 3K_r$. Putting all this together gives the desired bound.