Reducing Communication Overhead of the Subset Difference Scheme

Sanjay Bhattacherjee
Applied Statistics Unit, Indian Statistical Institute, Kolkata, 700108;
École Normale Supérieure de Lyon, Laboratoire LIP, 46 Allée d'Italie F69364 Lyon Cedex 07, France.
sanjay.bhattacherjee@gmail.com
and
Palash Sarkar
Applied Statistics Unit, Indian Statistical Institute, Kolkata, 700108.
palash@isical.ac.in

Abstract

In Broadcast Encryption (BE) systems like Pay-TV, AACS, online content sharing and broadcasting, reducing the header length (communication overhead per session) is of practical interest. The Subset Difference (SD) scheme due to Naor-Naor-Lotspiech (NNL) is the most popularly used BE scheme. We introduce the (a, b, γ) augmented binary tree subset difference ((a, b, γ)-ABTSD) scheme which is a generalization of the NNL-SD scheme. By varying the parameters (a, b, γ), it is possible to obtain $O(n \log n)$ different schemes. The average header length achieved by the new schemes is smaller than all known schemes having the same decryption time as that of the NNL-SD scheme and achieving non-trivial trade-offs between the user storage and the header size. The amount of key material that a user is required to store increases. For the earlier mentioned applications, reducing header size and achieving fast decryption is perhaps more of a concern than the user storage.

Keywords: Broadcast encryption, subset difference, binary trees, augmented trees, header length, transmission overhead, user storage, decryption time.

1 Introduction

Content protection in systems like Pay-TV, online media broadcasting and digital rights management [DRM] in optical discs, are based on broadcast encryption (BE) schemes. A symmetric key BE scheme assumes a broadcasting center and a set $\mathcal{N}$ of users who can receive the encrypted broadcast. At the outset, the users are provided long-lived keys which are stored on the user devices. Blocks of data are broadcast in sessions. For a given broadcast, a non-empty subset of users is identified as *privileged* and the other users are said to be *revoked*. Transmissions in each session are encrypted with a fresh random session key. The session key in turn is encrypted a number of times using a subset of long-lived keys such that each privileged user has access to one of these keys and no revoked user can access any of these keys. These encryptions of the session key are sent with the encrypted data block as the *header*. The number of times the session key is encrypted is called the *header length*. The communication overhead due to the scheme is measured by the header length.

The Advanced Access Content System [AAC] standard for digital rights management [DRM] in optical discs suggests the use of the Subset Difference (SD) scheme proposed in 2001 by Naor, Naor and Lotspiech [NNL01] which we will call the NNL-SD scheme. Consequent upon this standardisation, this scheme has become the most popularly used BE scheme. For a system with n users, the NNL-SD scheme requires each user to store $O(\log^2 n)$ secret key material. Further, for a broadcast having r revoked users, the worst case header length is

$2r-1$. Decryption time required by a user is $O(\log n)$ which is the fastest among all proposed schemes providing a non-trivial trade-off between the user storage and the header length.

1.1 Our Contributions

The starting point of our work is the NNL-SD scheme. For a system with $n = 2^{\ell_0}$ users, the NNL-SD scheme uses a full binary tree $\mathcal{T}^0$ of height ℓ_0 with the users as its leaves. The scheme identifies certain subsets of users and employs a clever strategy for assigning keys to these subsets. In any broadcast, the set of privileged users are covered using these subsets. A basic combinatorial intuition is that if we can somehow manage to increase the number of allowed subsets, then it may be possible to cover the privileged users using a smaller number of subsets. We follow up on this intuition.

Reducing the communication bandwidth can trivially be done if there is no restriction on the user storage. This, however, is an unrealistic scenario. We consider the issue of reducing communication bandwidth while at the same time ensuring that the increase in the user storage is not prohibitively expensive.

The new idea that we introduce uses the same underlying tree structure $\mathcal{T}^0$ as in the NNL-SD scheme and includes all the subsets that were assigned keys in the NNL-SD scheme. The additional subsets which are to be assigned keys are identified using small sub-graphs of $\mathcal{T}^0$ that are full binary trees of height a (≥ 1). These additional structures are called *a-trees*. Each internal node of $\mathcal{T}^0$ (at height greater than or equal to a) is augmented with an a-tree. Accordingly, the new scheme is called the a-augmented binary tree subset difference (a-ABTSD) scheme. For $a = 1$, the new scheme is the same as the NNL-SD scheme. For $a > 1$, the flexibility of having additional subsets arises by assigning keys to subsets of leaf nodes of each a-tree. As a result, the new scheme is a proper generalization of the NNL-SD scheme.

For a scheme with n users, the user storage is still $O(\log^2 n)$. The difference with the NNL-SD scheme is that the constant in the big-oh notation is proportional to 2^{k-1} where $k = 2^a$. So, for a fixed n, the a-ABTSD scheme is meaningful only if a is small. It is shown that the worst case header length of the a-ABTSD scheme is $2r-1$ (irrespective of the value of a) as in the case of the NNL-SD scheme. More importantly, we show that for any particular set of revoked users, the header size of the new scheme is never more than that of the NNL-SD scheme. Further, compared to the NNL-SD scheme, in the a-ABTSD scheme, the maximum header size of $2r-1$ is attained for larger values of n.

The main gain in using the a-ABTSD scheme is the reduction in the average header length. We have carried out experiments to study this. It turns out that for all values of r, the average header length of the new scheme for $a > 1$ is lower than that of the NNL-SD scheme. The lowering effect of the header length becomes more pronounced as either r increases or as a increases. Our results show that in scenarios where reducing communication bandwidth is a major concern, the new scheme provides an attractive alternative to the NNL-SD scheme.

We further propose two refinements to the a-ABTSD scheme. First, we restrict the cardinality (say b) of the subsets of leaves of a-trees that are assigned keys. Note that $b \in \{2, \ldots, 2^a - 1\}$. For $b = 2^a - 1$, the resulting scheme is the same as the a-ABTSD scheme. This refinement is used to mitigate the increase in the user storage while the (expected) header length stays better than the NNL-SD scheme. For $a \geq 2$ and $b \in \{2, \ldots, 2^a - 1\}$, we denote by (a, b)-ABTSD the scheme resulting from the consideration of the parameter b. The second refinement is obtained by introducing another parameter $a \leq \gamma \leq \ell_0$ to the (a, b)-aABTSD scheme which controls the maximum depth to which subset differences are allowed. This scheme is denoted as the (a, b, γ)-aABTSD scheme. For $\gamma = \ell_0$, the (a, b)-aABTSD scheme is obtained while for lower values of γ, the (a, b, γ)-aABTSD scheme has lower user storage and higher expected header length compared to the (a, b)-aABTSD scheme.

1.2 Comparison

There are three main parameters of a BE scheme – the header length, the user storage and the decryption time required by a privileged user. The new proposals have decryption time to be the same as that of the NNL-SD scheme, which is known to be the smallest among all schemes providing non-trivial trade-offs.

Among the schemes having the same decryption time as the NNL-SD scheme, there are two directions for trade-offs. One direction tries to reduce the user storage with a consequential increase in the header size and the other direction tries to reduce the header size with a consequential increase in the user storage. Applications of the first direction are resource constrained devices. The LSD scheme [HS02] and a subsequent work [BS14] have concentrated on the first direction. The trade-offs obtained in these works are not comparable to the present work.

The BDSD$_\lambda$ scheme from [WYL14, WYT13] is an example of the second direction and reduces the average header size at the cost of increasing the user storage. The (a, b, γ)-ABTSD scheme that we introduce provides a wide variety of trade-offs. In particular, we show that the $(\lambda, 2, \lambda)$-ABTSD scheme has average header size lower than that of the BDSD$_\lambda$ scheme. Expectedly, the user storage of the new schemes is more than that of the BDSD$_\lambda$ scheme. Details of the comparison and further comparison to the works [JHC$^+$05, HLL05] are provided in Section 7. To the best of our knowledge, there are no other works which reduces the header size of the NNL-SD scheme at the cost of increasing the user storage.

To summarise, among the schemes which have the same decryption time as that of the NNL-SD scheme, the schemes proposed here provide new non-trivial trade-offs where the average header size obtained using the new schemes is smaller than what could be achieved earlier. The resulting user storages are higher than previous schemes, but, would be tolerable for some applications. Such applications would include the Advanced Access Content System [AAC] for digital rights management [DRM] in optical disks and Pay-TV systems, where reducing the header size is the primary concern and a tolerable increase in key size would be acceptable.

1.3 Previous And Related Works

Broadcast encryption was introduced in [Ber91] and was formally studied in [FN93]. The work [FN93] proposed several schemes that were based on one-way functions and computational number-theoretic assumptions.

The subset cover framework for BE was proposed in [NNL01] that modelled almost all previously known BE schemes. The subset difference scheme, its formal security analysis and a traitor tracing technique were presented in this same work. Halevy and Shamir introduced the concept of layering in the underlying tree of the NNL-SD scheme and reduced the user storage [HS02].

Analysis of the expected header length of tree-based schemes in [NNL01, HS02] was done in [PB06]. It was shown in [EOPR08] that the standard deviations of the header lengths for these schemes are small compared to the means as the number of users gets large. Detailed worst case header length analysis for the tree-based schemes of [NNL01] have been done in [MMW09]. Efficient algorithms to compute the expected header lengths for these schemes were provided in [BS13, BS14] which also proposed several extensions of the NNL-SD and the LSD schemes.

A variant of the NNL-SD scheme was proposed in [GST04] which reduced the user storage to $O(\log n)$ at the cost of increasing the decryption time to $O(n)$. The works [JHC$^+$05, CJKY08] proposed BE schemes which were based on combinatorial structures different from that of a binary tree. A generic transformation to reduce user storage was proposed in [HLL05] and two concrete instantiations were provided. A modification of the NNL-SD scheme which reduces the transmission overhead was described in [WYL14, WYT13]. A ternary tree based subset difference technique was proposed in [FKTS08]. More recently, a generalization of the NNL-SD scheme to work for k-ary trees for any $k \geq 2$ was introduced in [BS15].

From a more combinatorial point of view, a trade-off between a given upper bound on the user storage and the consequent lower bound on the header length was given in [LS98] in the context of BE schemes for low-memory

stateless and low-state devices. Lower bounds on the header length for subset cover algorithms over different ranges of r were described in [AKI03].

We have considered the combinatorial framework of BE schemes. For applications such as the AACS and Pay-TV, the framework would be instantiated with symmetric key ciphers. BE schemes have also been studied extensively from the perspective of public-key cryptography. Many variants have been considered. These works are outside the scope of the present work and so we do not discuss them here.

2 Subset Cover Framework

Most known symmetric key BE schemes fall under the subset cover framework [NNL01]. In this framework, the broadcast center defines a collection $\mathcal{S}$ of subsets of the set of users $\mathcal{N}$ and keys are assigned to each subset in $\mathcal{S}$. For a user u, let $\mathcal{S}_u$ denote the subsets in $\mathcal{S}$ which contain u, i.e., $\mathcal{S}_u = \{S : S \in \mathcal{S} \text{ and } u \in S\}$. A user u gets secret information I_u from which it can derive the keys for all subsets in $\mathcal{S}_u$. This secret information I_u need not be the actual keys, instead it can consist of sufficient information which allows u to derive the key for any subset in $\mathcal{S}$ to which it belongs.

The message to be broadcast by the center in each session is encrypted with a random *session key* K_s. For each session, the center knows the set of revoked users $\mathcal{R}$. It forms a *partition* $\mathcal{S}_c$ of the set of privileged users $\mathcal{N} \setminus \mathcal{R}$ using subsets in $\mathcal{S}$, i.e., $\mathcal{S}_c \subseteq \mathcal{S}$; for $S_1, S_2 \in \mathcal{S}_c$, $S_1 \cap S_2 = \emptyset$; and $\cup_{S \in \mathcal{S}_c} S = \mathcal{N} \setminus \mathcal{R}$.

This set of subsets $\mathcal{S}_c$ is called the *subset cover* and the algorithm to find $\mathcal{S}_c$ is called the *cover generation or cover finding algorithm*. The session key is further encrypted for each subset $S \in \mathcal{S}_c$, using the key associated with S. These encryptions of the session key are sent as the header with the encrypted data. The number, h, of subsets in the cover $\mathcal{S}_c$, i.e., $h = |\mathcal{S}_c|$ is called the header length.

For decryption, a privileged user first determines the subset S in $\mathcal{S}_c$ to which it belongs. Then it uses the key corresponding to S to decrypt the portion of the header intended for S to obtain the session key K_s. Finally, it uses K_s to decrypt the message.

2.1 The Naor-Naor-Lotspiech Subset Difference Scheme

The subset difference scheme introduced by Naor, Naor and Lotspiech (NNL-SD) [NNL01] falls under the subset cover framework. The number of users n ($n = |\mathcal{N}|$) is assumed to be a power of two, i.e., $n = 2^{\ell_0}$ for some $\ell_0 \geq 0$.

A *full binary tree* $\mathcal{T}^0$ of height ℓ_0 forms the underlying structure for the scheme. Each user is associated with a unique leaf of $\mathcal{T}^0$. The nodes in the tree are numbered as follows. The root node is numbered 0. The left (resp. right) child of an internal node i is numbered $2i + 1$ (resp. $2i + 2$). For any node i in $\mathcal{T}^0$, the full binary tree rooted at i is denoted as $\mathcal{T}^i$.

There are a total of $\ell_0 + 1$ levels in the tree $\mathcal{T}^0$. The leaf nodes are at level 0; any internal node is at level $\ell + 1$ if its children are at level ℓ. So, the root node is at level ℓ_0. By $\mathsf{level}(i)$ we denote the level number of the node i in the tree $\mathcal{T}^0$. If J is a set of nodes all of which are at the same level, we will denote this common level by $\mathsf{level}(J)$.

Let i be a non-leaf node in $\mathcal{T}^0$ and j be a non-root node in $\mathcal{T}^i$. By $\mathcal{T}^i \setminus \mathcal{T}^j$ we denote the subgraph obtained by taking away $\mathcal{T}^j$ from $\mathcal{T}^i$. Let $S_{i,j}$ be the set of *leaf* nodes of $\mathcal{T}^i \setminus \mathcal{T}^j$.

2.1.1 The Collection NNL-$\mathcal{S}$

For the NNL-SD scheme, let us denote the collection of subsets which are assigned keys by NNL-$\mathcal{S}$. Then

$$\begin{aligned} \mathsf{NNL}\text{-}\mathcal{S} \;=\; & \{\mathcal{N}\} \cup \\ & \{S_{i,j} : i \text{ is a non-leaf node of } \mathcal{T}^0 \\ & \quad \text{and } j \text{ is a non-root node of } \mathcal{T}^i\}. \end{aligned} \tag{1}$$

The size of the collection NNL-$\mathcal{S}$ is $1 + \ell_0 2^{\ell_0+1} - 2^{\ell_0} + 1 = 2 + \ell_0 2^{\ell_0+1} - 2^{\ell_0}$.

2.1.2 PRG as a Hash function

Let m be the key-size of the underlying symmetric cipher and suppose that $\mathsf{PRG} : \{0,1\}^m \to \{0,1\}^{3m}$ is a cryptographic pseudo-random generator. On input L, let the output of $\mathsf{PRG}(L)$ be $R_0||R_1||R_2$ where R_t, $t = 0, 1, 2$, is an m-bit string. Let $G_t : \{0,1\}^m \to \{0,1\}^m$ be a function such that $G_t(L) = R_t$. A compact description of the functions G_0, G_1 and G_2 is as a hash function $G : \{0,1,2\} \times \{0,1\}^m \to \{0,1\}^m$ where $G_t(L) \stackrel{\Delta}{=} G(t, L)$. More generally, in this work, for a finite non-empty set F, a hash function $H : F \times \{0,1\}^m \to \{0,1\}^m$ is to be implemented using a pseudo-random generator $\mathsf{PRG} : \{0,1\}^m \times \{0,1\}^{mf}$ where $f = |F|$. We emphasize that the underlying cryptographic primitive is a PRG and the description in terms of the hash function is only for the sake of notational convenience.

2.1.3 Key Assignment To Subsets

A key K_0 is assigned to the subset $\mathcal{N}$. For key assignment to the other subsets in $\mathcal{S}$, a hash function $G : \{0,1,2\} \times \{0,1\}^m \to \{0,1\}^m$ (as mentioned above) is chosen by the center and is made available to all users in the system. Each subset $S_{i,j} \in \mathcal{S}$ is assigned a key as follows.

- Every internal node i in $\mathcal{T}^0$ is assigned a uniform random m-bit seed L_i.
- All non-root nodes j in the subtree $\mathcal{T}^i$ derive seeds from L_i in the following manner. Let $j = t_0, \ldots, t_p = i$ be the sequence of nodes in the path from j to i. Then for $\imath = p-1, \ldots, 0$, $t_\imath = 2t_{\imath+1} + s_\imath$ where $s_\imath \in \{1, 2\}$. Define the label $L_{i,j}$ associated to $S_{i,j}$ to be $L_{i,j} \stackrel{\Delta}{=} G_{s_0}(\cdots G_{s_{p-2}}(G_{s_{p-1}}(L_i)) \cdots)$.
- The key $K_{i,j}$ associated to the subset $S_{i,j}$ is defined to be $K_{i,j} \stackrel{\Delta}{=} G_0(L_{i,j})$.

2.1.4 The Set I_u For A User u

For a user u consider the set NNL-$\mathcal{S}_u$ of subsets in NNL-$\mathcal{S}$ which contain u. If $S_{i,j}$ is such a subset, then i is an ancestor of the leaf node u and j is not an ancestor of u. The user u should be able to generate the keys of all such subsets and no more. User u is at level 0 and suppose i is at level ℓ. Further suppose $u = i_0, i_1, \ldots, i_\ell = i$ be the path from u to i. Let $j_1, \ldots, j_\ell$ be the siblings of $i_1, \ldots, i_\ell$ respectively. Corresponding to the ancestor i at level ℓ, user u is given the ℓ seeds $L_{i,j_1}, \ldots, L_{i,j_\ell}$. Since u has ℓ_0 ancestors, the total number of seeds given to u is $\ell_0(\ell_0+1)/2$ plus the key K_0 assigned to the set $\mathcal{N}$. Denote the set of all seeds given to u by NNL-I_u, i.e.,

$$\begin{aligned} &\text{NNL-}I_u \\ &= \{K_0\} \cup \\ &\quad \{L_{i,j} : i \text{ is an ancestor of } u \text{ and } j \text{ is the sibling} \\ &\quad\ \text{of some node in the path from } u \text{ to } i\ \}. \end{aligned} \tag{2}$$

It can be seen that from the seeds that u gets, it can derive the keys for all subsets to which it belongs and no more.

3 The a-Augmented Binary Tree Subset Difference Scheme

The a-Augmented Binary Tree Subset Difference (a-ABTSD) scheme is a generalization of the NNL-SD scheme. It assumes an underlying full binary tree $\mathcal{T}^0$ as in the case of the NNL-SD scheme and imposes additional structure on this tree. The size of the structure is determined by a parameter a. For $a = 1$, the scheme turns out to be the same as the NNL-SD scheme.

3.1 Underlying Structure

As in the case of the NNL-SD scheme, there are $n = 2^{\ell_0}$ users associated with the leaves of the underlying full binary tree $\mathcal{T}^0$. The nodes and levels are also numbered as in the NNL-SD scheme.

For ease of later description, we introduce a few notions. Suppose J_1 and J_2 are two sets of nodes of $\mathcal{T}^0$ such that there is a node $j \in J_1$ and nodes $j_1, j_2 \in J_2$ such that $J_1 \setminus \{j\} = J_2 \setminus \{j_1, j_2\}$ and j_1, j_2 are the two children of j. Then the set J_2 can be thought of as being obtained from J_1 by replacing $\{j_1, j_2\}$ by j. Call the operation of replacing j_1, j_2 by their parent j to be a *moving-up* step.

Given a set J, it is possible to repeatedly apply the moving-up operation to get a set J' such that the moving-up operation can no longer be applied on J. We call J' to be a *reduced set*. Given a set J, there is a unique reduced set which can be obtained by repeatedly applying the moving-up step.

Let $\mathcal{T}$ be a full binary tree and J be a non-empty subset of the leaf nodes of $\mathcal{T}$. If J is either singleton, or, J can be reduced to a singleton set using moving-up operation, then J is called a *simple* subset of $\mathcal{T}$; otherwise, J is called a *non-simple* subset of $\mathcal{T}$. Figure 1 and Figure 2 show examples of simple and non-simple subsets respectively. By $\mathcal{J}_s(\mathcal{T})$ we denote the set of all simple subsets of $\mathcal{T}$. Similarly, $\mathcal{J}_{ns}(\mathcal{T})$ denotes the set of all non-simple subsets of $\mathcal{T}$. Note that both $\mathcal{J}_s(\mathcal{T})$ and $\mathcal{J}_{ns}(\mathcal{T})$ consist of subsets of the set of leaf nodes of $\mathcal{T}$.

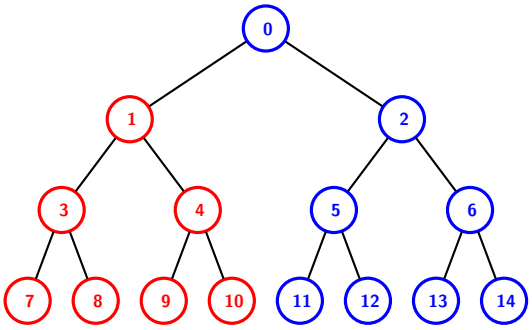


Figure 1: A full binary tree $\mathcal{T}$ with the set $J_1 = \{7, 8, 9, 10\}$ of leaf nodes that can be reduced to a singleton set $J_1' = \{1\}$. Hence, J_1 is a simple subset of $\mathcal{T}$.

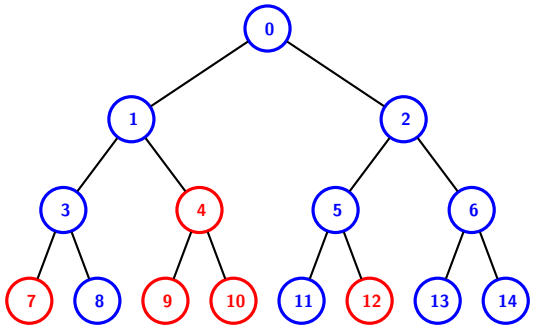


Figure 2: A full binary tree $\mathcal{T}$ where the set $J_2 = \{7, 9, 10, 12\}$ of leaf nodes may be reduced to $J_2' = \{7, 4, 12\}$ which is not singleton. Hence, J_2 is a non-simple subset of $\mathcal{T}$.

For the new scheme, additional structure is endowed to $\mathcal{T}^0$ in the following manner. Define an *a-tree* $\mathcal{A}_a^j$ to be a subgraph of $\mathcal{T}^0$ which is the full binary tree rooted at node j and of height a. So, the number of nodes in an a-tree is $1 + 2 + \ldots + 2^a = 2^{a+1} - 1$. The scheme is parameterized by the number a.

We provide an example to illustrate this notion. In Figure 3 where $a = 2$, the subtree rooted at node 4 is the a-tree $\mathcal{A}_2^4$ containing the nodes $\{4, 9, 10, 19, 20, 21, 22\}$. Another a-tree $\mathcal{A}_2^1$ is the subgraph containing the nodes $\{1, 3, 4, 7, 8, 9, 10\}$.

For a fixed value of a in $\mathcal{T}^0$, each a-tree is uniquely identified by its root node. Alternatively, suppose J is a non-empty subset of leaf nodes of an a-tree $\mathcal{A}_a^j$ such that the nodes in J are at level ℓ (of $\mathcal{T}^0$). Then the root j is the unique ancestor at level $\ell + a$ of the nodes in J. So, given J, the node j is uniquely determined and we will call j to be the *a-pivot* of J.

The level number of the root node of any a-tree in $\mathcal{T}^0$ is at least a. Hence, for a full binary tree with $n = 2^{\ell_0}$ leaves, the number of distinct a-trees is the number of internal nodes at levels between ℓ_0 and a. Since there are $2^{\ell_0 - \ell}$ nodes at level ℓ in $\mathcal{T}^0$, hence the number of a-trees is

$$1 + 2 + \ldots + 2^{\ell_0 - a} = 2^{\ell_0 - a + 1} - 1.$$

For any internal node i of $\mathcal{T}^0$ and any non-root node j in $\mathcal{T}^i$, $\mathcal{T}^i \setminus \mathcal{T}^j$ is the subgraph of $\mathcal{T}^i$ obtained by taking away $\mathcal{T}^j$. We generalize this notion in the following manner. As before, let i be a non-leaf node in $\mathcal{T}^0$ and let $J = \{j_1, \ldots, j_c\}$ be a non-empty subset of non-root nodes in $\mathcal{T}^i$. Define $\mathcal{T}_{i,J}$ to be the subgraph of $\mathcal{T}^i$ formed by taking away all of $\mathcal{T}^{j_1}, \ldots, \mathcal{T}^{j_c}$ from $\mathcal{T}^i$. In other words,

$$\mathcal{T}_{i,J} = \mathcal{T}^i \setminus \left(\mathcal{T}^{j_1} \cup \cdots \cup \mathcal{T}^{j_c}\right).$$

Let $S_{i,J}$ denote the set of leaf nodes of the subgraph $\mathcal{T}_{i,J}$.

Suppose J_1 and J_2 are two sets of nodes in $\mathcal{T}^i$ such that J_2 is obtained from J_1 by a moving-up step. Then it is easy to see that the set of leaf nodes of $\mathcal{T}_{i,J_1}$ is the same as the set of leaf nodes of $\mathcal{T}_{i,J_2}$ and so $S_{i,J_1} = S_{i,J_2}$. We say (i, J_1) and (i, J_2) are two representations of the set $S_{i,J_1} = S_{i,J_2}$. If J' is a reduced set obtained by successively applying the moving-up operation to a set J, then $S_{i,J} = S_{i,J'}$. By an extension of terminology, we will call the representation (i, J') to be the reduced form representation of the set $S_{i,J}$.

3.2 The Collection $\mathcal{S}$

Let i be an internal node of $\mathcal{T}^0$ and J be a non-simple subset of $\mathcal{A}_a^j$ where j is a node of $\mathcal{T}^i$. We call such a pair (i, J) to be *allowed*.

Suppose (i, J) is an allowed pair where the nodes in J are at level ℓ. Then the level of the a-pivot j of J is $\ell + a$ and so the level of i is at least $\ell + a$. This shows that there cannot be an allowed pair (i, J) where the level of i is less than a.

The collection $\mathcal{S}$ consists of the following subsets:

- all NNL-SD subsets $S_{i,j}$; and
- $S_{i,J}$ for all allowed pairs (i, J).

In other words,

$$\mathcal{S} \quad = \quad \mathsf{NNL}\text{-}\mathcal{S} \cup \mathcal{A}\text{-}\mathcal{S} \tag{3}$$

where $\mathcal{A}\text{-}\mathcal{S} \triangleq \{S_{i,J} : (i, J) \text{ is allowed}\}$.

For $S_{i,J} \in \mathcal{A}\text{-}\mathcal{S}$, J is non-simple and so J cannot be reduced to a singleton set using moving-up operations. As a result, $S_{i,J}$ is not equal to any NNL-SD subset. So, the collections $\mathsf{NNL}\text{-}\mathcal{S}$ and $\mathcal{A}\text{-}\mathcal{S}$ are disjoint.

If $a = 1$, then any J which is a non-empty subset of the leaf nodes of an a-tree is necessarily simple. So, there are no allowed pairs (i, J) showing that $\mathcal{A}\text{-}\mathcal{S} = \emptyset$. As a consequence, in this case, the a-ABTSD scheme collapses to the NNL-SD scheme.

As an example, let us consider the tree $\mathcal{T}^0$ in Figure 3 with 16 users. It shows the subset that has been formed by excluding the users in $\mathcal{T}^7$, $\mathcal{T}^9$ and $\mathcal{T}^{10}$ from the users in $\mathcal{T}^0$. The subset is denoted as $S_{0,\{7,9,10\}}$. Nodes $\{7, 9, 10\}$ are leaves of the a-tree $\mathcal{A}_2^1$. Note that the set $\{7, 4\}$ can be obtained from the set $\{7, 9, 10\}$ by a moving-up operation. So, $S_{0,\{7,9,10\}} = S_{0,\{7,4\}}$.

3.3 Key Assignment To Subsets In $\mathcal{S}$

The key assignment strategy is an extension of the strategy for the NNL-SD scheme. The collection $\mathcal{S}$ consists of two sub-collections $\mathsf{NNL}\text{-}\mathcal{S}$ and $\mathsf{A}\text{-}\mathcal{S}$. We assume as in the case of the NNL-SD scheme that each internal node i of $\mathcal{T}^0$ is assigned an independent and uniform random m-bit seed L_i. Further, for any non-root j in $\mathcal{T}^i$, the seed $L_{i,j}$ is also defined using G_t as in the NNL-SD scheme and the key for the NNL-SD subset $S_{i,j}$ is $K_{i,j} = G_0(L_{i,j})$. In other words, keys to the subsets in $\mathsf{NNL}\text{-}\mathcal{S}$ are assigned as in the NNL-SD scheme. For convenience of notation, we define $L_{i,i} \triangleq L_i$.

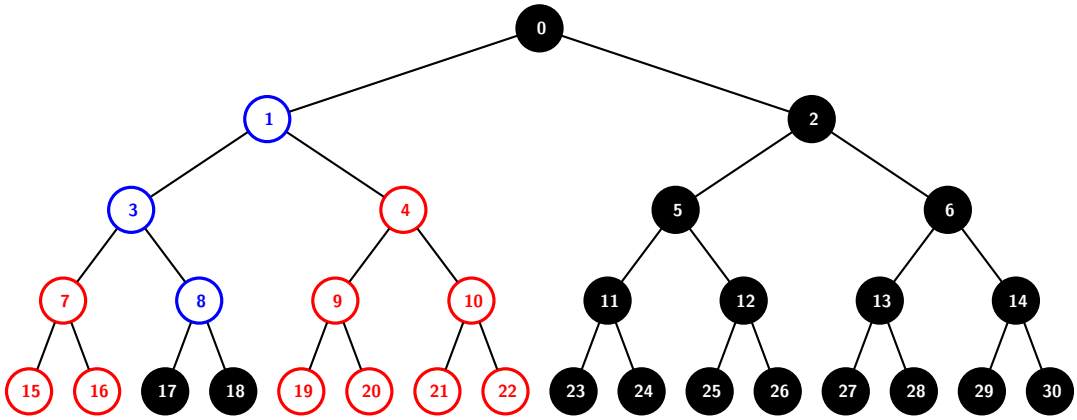


Figure 3: The binary tree $\mathcal{T}^0$ that is the underlying structure of the a-ABTSD scheme for $n = 16$ users is shown here. The red leaf nodes denote revoked users while the black ones denote privileged users. Here we assume $a = 2$. The subset $S_{0,\{7,9,10\}} = \{17, 18, 23, 24, \ldots, 30\}$ from the collection $\mathcal{S}$ ($\mathcal{A}$-$\mathcal{S}$ in particular) is also shown. It has all users in the subtree $\mathcal{T}^0$ but not in $\mathcal{T}^7 \cup \mathcal{T}^9 \cup \mathcal{T}^{10}$. Since $J = \{7, 9, 10\}$ is a non-simple subset of the a-tree $\mathcal{A}_2^1$, $(1, J)$ is an allowed pair. Using the moving up operation, the subset J may also be represented as $S_{0,\{7,4\}}$.

Let $\mathcal{T}$ be a full binary tree of height a and as defined earlier $\mathcal{J}_{ns}(\mathcal{T})$ is the set of all non-simple subsets of $\mathcal{T}$. Following the description in Section 2.1.2, we use a PRG to define a hash function

$$H[\mathcal{T}] : \mathcal{J}_{ns}(\mathcal{T}) \times \{0,1\}^m \rightarrow \{0,1\}^m. \tag{4}$$

Keys to the subsets in $\mathcal{A}$-$\mathcal{S}$ are defined using the hash function H. Note that H is defined with respect to the tree $\mathcal{T}$. This is because the domain of H depends on $\mathcal{T}$. On the other hand, we expect H to act on any full binary tree of height a in the same manner. So, when $\mathcal{T}$ is clear from the context, we will write H instead of $H[\mathcal{T}]$.

Let $k = 2^a$ which is the number of leaf nodes in any a-tree. Suppose $S_{i,J}$ is in the collection $\mathcal{A}$-$\mathcal{S}$. Then (i, J) is an allowed pair and suppose the a-pivot of J is j. Then J is necessarily a non-simple subset of $\mathcal{A}_a^j$, i.e., $J \in \mathcal{J}_{ns}(\mathcal{A}_a^j)$. *The key $K_{i,J}$ assigned to $S_{i,J}$ is*

$$K_{i,J} \triangleq H[\mathcal{A}_a^j](J, L_{i,j}). \tag{5}$$

Note that j can be equal to i and in that case $L_{i,i}$ is simply L_i.

3.4 Number Of Subsets In The Collection

As mentioned earlier, the count of the number of NNL-SD subsets is $2 + \ell_0 2^{\ell_0+1} - 2^{\ell_0}$. We now consider the number of subsets in $\mathcal{A}$-$\mathcal{S}$. The following result gives the number of simple and non-simple subsets of a full binary tree of height a.

Lemma 1. *Let $\mathcal{T}$ be a full binary tree of height a and $k = 2^a$. Then the number of simple subsets of $\mathcal{T}$, i.e. $|\mathcal{J}_s(\mathcal{T})|$ equals $2k - 1$. Consequently, the number of non-simple subsets of $\mathcal{T}$, i.e. $|\mathcal{J}_{ns}(\mathcal{T})|$, equals $2^k - 2k$.*

Proof. $\mathcal{T}$ has $k = 2^a$ leaf nodes and a total of $2k - 1$ nodes. If J is a simple subset of $\mathcal{T}$, then J is either a singleton subset of the set of leaf nodes of $\mathcal{T}$ or, can be reduced to one of the internal nodes of $\mathcal{T}$. So, the number of simple nodes of $\mathcal{T}$ is $2k - 1$. The total number of non-empty subsets of the leaf nodes of $\mathcal{T}$ is $2^k - 1$. Out of these $2k - 1$ are simple subsets. As a result, there are $2^k - 2k$ non-simple subsets of $\mathcal{T}$. □

Fix a node i of $\mathcal{T}^0$ with $\mathsf{level}(i) = \ell$. Out of the $2^{\ell+1} - 1$ nodes in $\mathcal{T}^i$, $2^{\ell-a+1} + \ldots + 2^{\ell}$ nodes are at the bottom-most a levels. These nodes cannot be the a-pivot for any set J such that the pair (i, J) is allowed. Each of the remaining $2^{\ell-a+1} - 1$ nodes in $\mathcal{T}^i$ will be the root of an a-tree that generate subsets. For a node i, each such a-tree will generate $2^k - 2k$ subsets of the form $S_{i,J}$ where J is non-simple. Thus, the total number of subsets of the form $S_{i,J}$ in $\mathcal{A}$-$\mathcal{S}$ is

$$\begin{aligned}&\sum_{\ell=a}^{\ell_0} 2^{\ell_0-\ell}(2^{\ell-a+1} - 1)(2^k - 2k - 2)\\ &= (2^k - 2k)((\ell_0 - a)2^{\ell_0-a+1} - 2^{\ell_0-a+1} + 1).\end{aligned}$$

Hence, the total number of subsets in the collection $\mathcal{S}$ is

$$\begin{aligned}|\mathcal{S}| &= |\mathsf{NNL}\text{-}\mathcal{S}| + |\mathcal{A}\text{-}\mathcal{S}|\\ &= 2 + \ell_0 2^{\ell_0+1} - 2^{\ell_0}\\ &\quad +(2^k - 2k)((\ell_0 - a)2^{\ell_0-a+1} - 2^{\ell_0-a+1} + 1).\end{aligned} \tag{6}$$

3.5 I_u Per User u

Let u be a user, i.e. a leaf node of $\mathcal{T}^0$. The information provided to u consists of two disjoint subsets which we call $I_u^{(1)}$ and $I_u^{(2)}$.

3.5.1 The Subset $I_u^{(1)}$

The first part is the same as that in the NNL-SD scheme, i.e., $I_u^{(1)} = \mathsf{NNL}\text{-}I_u$. Recall that $\mathsf{NNL}\text{-}I_u$ consists of seeds $L_{i,j}$ where i is an ancestor of u and j is the sibling of some node in the path from u to i. As mentioned earlier, the number of m-bit seeds in $I_u^{(1)}$ is $|I_u^{(1)}| = 1 + \ell_0(\ell_0 + 1)/2$. From the seeds in $I_u^{(1)}$, u can derive keys of the following type:

- key $K_{i,j}$ corresponding to any NNL-SD subset $S_{i,j}$ containing u;
- key $K_{i,J}$ corresponding to any subset $S_{i,J}$ containing u such that the a-pivot of J is in the subtree rooted at the sibling of some node in the path from u to i.

The seeds in $I_u^{(1)}$ are not actual keys for subsets. These actual keys have to be derived from the seeds by one or more applications of the hash functions G and/or H. Following Section 2.1.2, the applications of the hash functions are actually applications of the underlying PRG.

3.5.2 The Subset $I_u^{(2)}$

Let $\mathcal{T}$ be a full binary tree of height a and v be a leaf node of $\mathcal{T}$. Let $\mathcal{J}_{ns,v}(\mathcal{T})$ denote the set of all non-simple sets of $\mathcal{T}$ not containing v. In other words, J is in $\mathcal{J}_{ns,v}(\mathcal{T})$ if J is a non-empty subset of the leaf nodes of $\mathcal{T}$, J cannot be reduced to singleton subset and $v \notin J$.

Lemma 2. *Let $\mathcal{T}$ be a full binary tree of height a and v be a leaf node of $\mathcal{T}$. Then $|\mathcal{J}_{ns,v}(\mathcal{T})| = 2^{k-1} - 2k + a + 1$.*

Proof. Consider a non-empty subset of the leaf nodes of $\mathcal{T}$ not containing v. Since $\mathcal{T}$ has k leaf nodes, there are a total of $2^{k-1} - 1$ possibilities for J. Further J cannot be reduced to any of the ancestors of v in $\mathcal{T}$. □

Define $\mathcal{S}_u^{(2)}$ to be collection of subsets $S_{i,J}$ in $\mathcal{A}$-$\mathcal{S}$ satisfying the following conditions:

- i is an ancestor of u and the a-pivot j of J is also an ancestor of u;
- the ancestor v of u at $\mathsf{level}(J)$ is not in J.

Define

$$I_u^{(2)} \quad = \quad \{K_{i,J} : S_{i,J} \text{ is in } \mathcal{S}_u^{(2)}\}. \tag{7}$$

The size of $I_u^{(2)}$ is calculated as follows. If i is at level ℓ, then the possible levels for the a-pivot j of J are $a, a+1, \ldots, \ell$. Fix a level ℓ' of j. We now need to find the number of non-simple subsets J satisfying the above conditions. There are $k = 2^a$ leaf nodes of $\mathcal{A}_a^j$. The ancestor v of u at level ℓ' is a leaf node of $\mathcal{A}_a^j$. By the above condition, v should not be in J and so there are $k-1$ leaf nodes of $\mathcal{A}_a^j$ which can be in J. Any subset J' of the leaf nodes of $\mathcal{A}_a^j$ which does not contain v cannot be reduced to any of the singleton nodes in the path from v to j (both inclusive). There are a total of $(2k-1)-(a+1)$ nodes in $\mathcal{A}_a^j$ to which it may be possible to reduce J' by applying moving-up operations. So, the number of J satisfying the required conditions is $2^{k-1}-1-(2k-a-2)$. For a node i at level ℓ, there are $(\ell - a + 1)$ possible choices for j and for each j there are $2^{k-1} - 2k + a + 1$ choices for J. So, the number of keys in $I_u^{(2)}$ is

$$\begin{aligned} |I_u^{(2)}| \quad &= \quad \sum_{\ell=a}^{\ell_0} (\ell - a + 1)(2^{k-1} - 2k + a + 1) \\ &= \quad \frac{1}{2} \times (2^{k-1} - 2k + a + 1) \\ &\qquad \times (\ell_0 - a + 2)(\ell_0 - a + 1). \end{aligned} \tag{8}$$

Recall that for a user u, $\mathcal{S}_u$ denotes the collection of subsets in $\mathcal{S}$ which contain u. Also, $\mathsf{NNL}\text{-}\mathcal{S}_u$ denotes the collection of all NNL-SD subsets which contain u. Define $\mathcal{A}\text{-}\mathcal{S}_u$ to be the collection of all subsets from $\mathcal{A}\text{-}\mathcal{S}$ which contain u. Then $\mathcal{S}_u$ is the disjoint union of $\mathsf{NNL}\text{-}\mathcal{S}_u$ and $\mathcal{A}\text{-}\mathcal{S}_u$. The set $I_u^{(1)}$ provides u with information to generate keys for any subset in $\mathsf{NNL}\text{-}\mathcal{S}_u$. Similarly, the set $I_u^{(2)}$ provides u with information to generate keys for any subset in $\mathcal{A}\text{-}\mathcal{S}_u$. Further, the two sets $I_u^{(1)}$ and $I_u^{(2)}$ are disjoint and their union is the set I_u which provides u with information to generate keys for any subset in $\mathcal{S}_u$. The total number of m-bit seeds that u needs to store is the cardinality of I_u and is given by the following.

$$\begin{aligned} |I_u| \quad &= \quad |I_u^{(1)}| + |I_u^{(2)}| \\ &= \quad 1 + \frac{\ell_0(\ell_0+1)}{2} \\ &\qquad + \frac{(2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1)}{2}. \end{aligned} \tag{9}$$

For a fixed k and as n grows, the expression in (9) is $O(\log^2 n)$ which is the same as that of the NNL-SD scheme. This is much better than the number of keys being proportional to n. On the other hand, for a fixed n as k increases, the number of keys also increases. The set $I_u^{(2)}$ consists of actual keys for the subsets in $\mathcal{S}_u^{(2)}$.

In the supplementary material we show how to define the hash function H such that the definition of $I_u^{(2)}$ can be altered to provide information using which seeds in $\mathcal{S}_u^{(2)}$ can be derived. This results in decreasing the factor $(2^{k-1} - 2k + a + 1)$ in the above expression.

Suppose the number of users is n. Then as discussed earlier, the user storage is not the same for all users. Denote by $\mathsf{us}_a(n)$ the maximum user storage with n users, i.e., $\mathsf{us}_a(n) = \max_u |I_u|$. For $2^{\ell_0 - 1} < n \leq 2^{\ell_0}$, $\mathsf{us}_a(n) = \mathsf{us}_a(2^{\ell_0})$. For $a = 2$, by direct considerations we are able to show the following.

$$\mathsf{us}_2(2^{\ell_0}) \quad = \quad 1 + \ell_0(\ell_0+1)/2 + \ell_0(\ell_0-1). \tag{10}$$

The case for arbitrary a needs a more general approach. Using a cyclotomic coset based technique from [BS15], we are able to show the following.

$$\begin{aligned} \mathsf{us}_a(2^{\ell_0}) &= 1 + \frac{\ell_0(\ell_0+1)}{2} \\ &\quad + \frac{(\chi_{2^a} - 2) \times (\ell_0 - a + 2)(\ell_0 - a + 1)}{2}. \end{aligned} \tag{11}$$

Here χ_k is the number of cyclotomic cosets of k-bit strings. Details of how this can be done is discussed in the supplementary material.

4 Cover Finding Algorithm

The algorithm takes as input the set $\mathcal{R}$ of revoked users and outputs the subset cover $\mathcal{S}_c$. If $\mathcal{R} = \emptyset$ then the only set in the subset cover is the set $\mathcal{N}$ of all users. If $\mathcal{R} \neq \emptyset$, then the subset cover consists of NNL-SD subsets $S_{i,j}$ or $S_{i,J}$ for allowed pairs (i, J). The subset cover algorithm that we describe below identifies NNL-SD subsets $S_{i,j}$ with $S_{i,\{j\}}$. For any allowed pair (i, J), the algorithm obtains $S_{i,J'}$ where J' is the reduced form of J.

The algorithm runs iteratively and maintains a list $\mathcal{L}$ of nodes on the paths joining revoked leaf nodes with the root. The list $\mathcal{L}$ is initially populated with the revoked leaf nodes, all marked as covered. The algorithm runs from left to right on this list and keeps adding the parent nodes of each node in the list until the root. Each node j in the list has an associated list $\mathsf{SDnodes}[j]$ of its descendant nodes. For a node j at level $\mathsf{level}(j) \geq a$, the nodes in $\mathsf{SDnodes}[j]$ are in an a-tree rooted at j or at some descendant of j. For a node j at level $\mathsf{level}(j) < a$, the list $\mathsf{SDnodes}[j]$ will have nodes from the subtree $\mathcal{T}^j$. While investigating the child nodes of i in the list, $\mathsf{SDnodes}[i]$ and the status of i are updated. The algorithm works as follows.

Algorithm $\mathcal{C}$. Takes as input the set $\mathcal{R} \neq \emptyset$ of revoked users and outputs the subset cover $\mathcal{S}_c$. Each subset in $\mathcal{S}_c$ is in reduced form.

1. Form the initial list $\mathcal{L}$ with all revoked leaf nodes of $\mathcal{T}^0$. Mark each node j as covered and set $\mathsf{SDnodes}[j] = \{j\}$. Set $\mathcal{S}_c$ to be the empty set.

2. Process nodes in $\mathcal{L}$ from left to right. Let $\mathcal{L}[t]$ be the node that is processed at the t^{th} iteration. If $\mathcal{L}[t]$ is the root node, go to step 3. Let i be the parent of $\mathcal{L}[t]$. At the t^{th} iteration:

   (a) If $\mathcal{L}[t]$ and $\mathcal{L}[t+1]$ have the same parent, proceed to the next iteration for $\mathcal{L}[t+1]$.

   (b) Else, append i to $\mathcal{L}$. Node i can have at most two children in $\mathcal{L}$. Let the children of i in $\mathcal{L}$ be $\{j_1, j_d\}$ where $(1 \leq d \leq 2)$. The following mutually exclusive cases occur:

      i. Case when all d children of i are covered:
         A. If $d = 1$, mark i as intermediate and set $\mathsf{SDnodes}[i] = \{j_1\}$.
         B. For $d = 2$, mark i as covered and set $\mathsf{SDnodes}[i] = \{i\}$.

      ii. Case when $d = 1$ and j_1 is intermediate:
      Mark i as intermediate and copy $\mathsf{SDnodes}[j_1]$ to $\mathsf{SDnodes}[i]$.

      iii. Case when $d = 2$ and at least one node in $\{j_1, j_2\}$ is intermediate:
         A. If for some $j \in \{j_1, j_2\}$, there is a $j' \in \mathsf{SDnodes}[j]$ such that $\mathsf{level}(j) - \mathsf{level}(j') \geq a$, then for each $j \in \{j_1, j_2\}$ that is marked as intermediate, add $S_{j,\mathsf{SDnodes}[j]}$ to $\mathcal{S}_c$. Subsequently, mark i as covered and set $\mathsf{SDnodes}[i] = \{i\}$.
         B. Otherwise, mark i as intermediate and set $\mathsf{SDnodes}[i]$ to $\mathsf{SDnodes}[j_1] \cup \mathsf{SDnodes}[j_2]$.

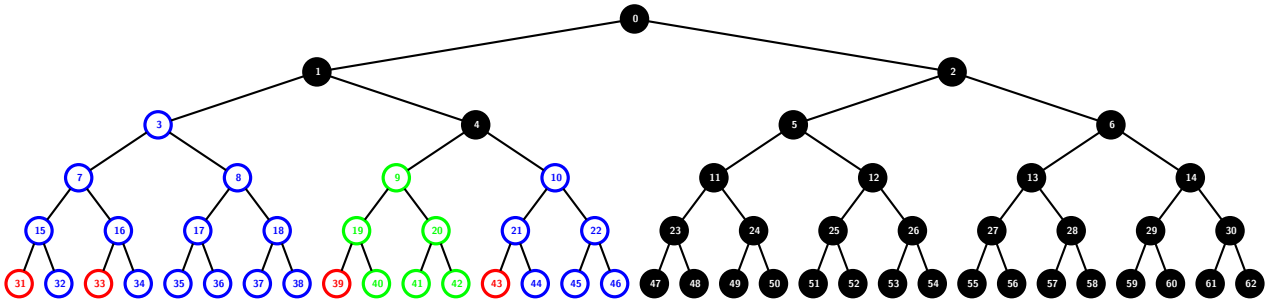


Figure 4: Example of a subset cover for $\mathcal{R} = \{31, 33, 39, 43\}$ in the a-ABTSD scheme with $a = 2$ and $n = 32$ users. The subsets in the cover are $S_{3,\{31,33\}}$, $S_{9,\{39\}}$, $S_{10,\{43\}}$ and $S_{0,\{1\}}$.

3. If the root node is marked as intermediate, add $S_{0,\mathsf{SDnodes}[0]}$ to the cover $\mathcal{S}_c$.

The subset cover $\mathcal{S}_c$ output by the algorithm is a collection of subsets of the form $S_{i,\mathsf{SDnodes}[i]}$.

Figure 4 shows an example where $a = 2$, $n = 32$ and $\mathcal{R} = \{31, 33, 39, 43\}$. Hence, the list $\mathcal{L}$ eventually gets populated with the nodes $\{31, 33, 39, 43, 15, 16, 19, 21, 7, 9, 10, 3, 4, 1, 0\}$ that lie on the paths joining the revoked leaves with the root node. The subsets generated by the algorithm working on the above list are $S_{9,\{39\}}$, $S_{10,\{43\}}$, $S_{3,\{31,33\}}$ and $S_{0,\{1\}}$.

The cover finding algorithm may be understood as finding an NNL-SD cover and in the process, combining those subsets in the cover whose union occurs in $\mathcal{A}$-$\mathcal{S}$, i.e., the union is a subset which arises due to the additional a-tree that has been added to the full binary tree. The correctness of the cover finding algorithm is provided in the supplementary material. It is also shown that the maximum header length of the a-ABTSD scheme for $a > 1$ is never more and in general less than that of the NNL-SD scheme. Further, the maximum header length is shown to be $2r - 1$ and it is also proved that this bound is tight.

4.1 Decryption Time

Each user is at a leaf of $\mathcal{T}^0$ which is a full binary tree having n leaf nodes and hence has height $\log n$. In a broadcast, each privileged user u has to use information present in I_u to derive a key corresponding to the subset of the broadcast to which it belongs. Such information is essentially an m-bit intermediate seed and the process of deriving a key consists of repeatedly applying the proper hash function. For any intermediate seed, at most $\log n$ applications of the hash function lead to a key. So, u can derive the proper key in $O(\log n)$ time. This time is the same as that required for the NNL-SD scheme.

5 The (a, b)-ABTSD Refinement

The a-ABTSD scheme is parameterised by the height $a \in \{1, \ldots, \ell_0\}$ of the additional a-tree structure where $\ell_0 = \log n$ is the height of the tree $\mathcal{T}^0$. The two ends of the trade-off determined by a are the following.

1. At one end, for $a = 1$ the scheme is identical to the NNL-SD scheme.

2. At the other end, for $a = \log n = \ell_0$ the collection $\mathcal{S}$ is the collection of all non-empty subsets of $\mathcal{N}$. This is called the *power-set scheme* where, given any set R of revoked users, the set of privileged users $\mathcal{N} \setminus \mathcal{R}$ is in $\mathcal{S}$ and hence has a key assigned to it. The subset cover $S_c = \{\mathcal{N} \setminus \mathcal{R}\}$ and hence the header length will be 1. The user storage, on the other hand, will be $O(\chi_n \log^2 n)$ (which is exponential in n) as can be seen from (11).

For intermediate values of $a \in \{2, \ldots, \ell_0 - 1\}$, there are $\ell_0 - 2 = O(\log n)$ different storage/header-length trade-off points which lie between the NNL-SD scheme and the power set scheme. As the value of a increases, an increasing number of subsets are assigned keys and hence the expected header length reduces while the user storage increases.

This hierarchy of trade-offs can be further refined as we now explain. First consider a fixed value of a. We introduce an *additional parameter* $b \in \{2, \ldots, 2^a - 1\}$ to restrict the number of subsets that are assigned keys.

The a-tree $\mathcal{A}_a^j$ is defined as before to be the subgraph of $\mathcal{T}^0$ that is a full binary tree of height a rooted at node j. We recollect from Section 3 that a subset J of leaf nodes of $\mathcal{A}_a^j$ is said to be simple if it can be reduced to a single node by the moving-up operation. Let i be an internal node of $\mathcal{T}^0$ and J be a non-simple subset of leaves of $\mathcal{A}_a^j$ where j is a non-leaf node of $\mathcal{T}^i$.

We call a pair (i, J) to be b-*allowed* if J is non-simple and $|J| \leq b$. For $a > 1$, we define the (a, b)-ABTSD scheme where $\mathcal{S}$ is defined to be:

$$\mathcal{S} \quad = \quad \mathsf{NNL}\text{-}\mathcal{S} \cup \mathcal{A}\text{-}\mathcal{S} \tag{12}$$

where $\mathcal{A}\text{-}\mathcal{S} \triangleq \{S_{i,J} : (i, J) \text{ is } b\text{-allowed}\}$.

For $a = 1$, we still have only the NNL-SD scheme; for $2 \leq a \leq \ell_0$, the parameter b in the (a, b)-ABTSD scheme can take any of the $2^a - 2$ values in the set $\{2, \ldots, 2^a - 1\}$. The $(a, 2^a - 1)$-ABTSD scheme is just the a-ABTSD scheme. All possible choices of a and b give rise to a total of $\sum_{a=1}^{\ell_0}(2^a - 2) = 2^{\ell_0+1} - 2\ell_0 - 1 = O(n)$ possible schemes having different storage/header-length trade-off points which lie between the NNL-SD scheme and the power set scheme. For a fixed a, as b increases the number of subsets in the collection increases leading to an increase in the user storage and a decrease in the (expected) header length. Note that the worst case header length for all intermediate schemes is $2r - 1$ (same as NNL-SD) where r is the number of revoked users. As the parameter values are increased, the probability of occurrence of the worst case header length never increases and decreases in general.

The cover generation algorithm $\mathcal{C}$ (in Section 4) of the a-ABTSD scheme can be modified to obtain the cover generation algorithm for the (a, b)-ABTSD scheme. This modification consists of modifying only Step 2)-b)-iii)-A) of Algorithm $\mathcal{C}$ in the following manner. No other step of Algorithm $\mathcal{C}$ needs to be changed.

C1:= (for some $j \in \{j_1, j_2\}$*, there is a* $j' \in \mathsf{SDnodes}[j]$ *such that* $\mathsf{level}(j) - \mathsf{level}(j') \geq a$*).*

C2:= $(\sum_{j' \in S} |\mathsf{leaves}(i, j')| > b)$*.*

If conditions C1 or C2 (or both) are true, then for each $j \in \{j_1, j_2\}$ *that is marked as* ***intermediate****, add* $S_{j,\mathsf{SDnodes}[j]}$ *to* $\mathcal{S}_c$ *and subsequently mark* i *as* ***covered*** *and set* $\mathsf{SDnodes}[i] = \{i\}$*.*

Here $S = \mathsf{SDnodes}[j1] \cup \mathsf{SDnodes}[j2]$, $\mathsf{leaves}(u, v)$ denotes the leaf nodes of the a-tree rooted at node u that are in the subtree rooted at node v and $|\mathsf{leaves}(u, v)| = 2^{a-(\mathsf{level}(u)-\mathsf{level}(v))}$.

The first condition in the disjunct of the 'if' condition above is the same as that for the a-ABTSD scheme. Also, the method of adding subsets to the cover remains unchanged. The only change is the introduction of the condition based on the parameter b. This condition captures the requirement that in a b-allowed pair (i, J), the cardinality of J is at most b. So, if the cardinality of the current J $(= \cup_{j' \in S} \mathsf{leaves}(i, j'))$ is more than b, then subsets are added to the cover.

The key assignment method for the (a, b)-ABTSD scheme remains almost the same as that of the a-ABTSD scheme with the only difference being the fact that keys are assigned only to subsets represented by b-allowed pairs (i, J). For user storage, consider the expression given by (9). The factor $(2^{k-1} - 2k + a + 1)$ (where $k = 2^a$) in (9) counts the number of non-simple subsets J of the leaf nodes of an a-tree which do not contain a fixed leaf node of the same a-tree. For the restricted (a, b)-ABTSD scheme, we have to replace this factor by $\xi_{a,b}$ which counts the number of non-simple subsets J of cardinality at most b of the leaf nodes of an a-tree which do not contain a fixed leaf node of the same a-tree. It is complicated to find the formula for $\xi_{a,b}$, but, an easy upper bound is $\xi_{a,b} \leq \sum_{t=1}^{b} \binom{k-1}{t}$. Replacing $(2^{k-1} - 2k + a + 1)$ by $\xi_{a,b}$ in (9) provides the user storage for the (a, b)-ABTSD scheme to be the following: $1 + \ell_0(\ell_0 + 1) + \xi_{a,b}(\ell_0 - a + 1)(\ell_0 - a + 2)/2$.

5.1 The (a, b, γ)-ABTSD Scheme

Consider the (a, b)-ABTSD scheme with $a > 1$. The collection $\mathcal{S}$ consists of the NNL-SD subsets along with additional subsets of the type $S_{i,J}$ with (i, J) being b-allowed. Note that the condition for b-allowed requires the a-pivot of J to be a node in $\mathcal{T}^i$.

It is possible to define a restricted variant of the (a, b)-ABTSD scheme. In the restricted variant, the additional subsets are of the type $S_{i,J}$ with (i, J) being b-allowed *and additionally,* the distance between i and the nodes in J is at most a pre-determined value γ ($a \leq \gamma \leq \ell_0$). We denote this variant as the (a, b, γ)-ABTSD scheme.

For a node i in $\mathcal{T}^0$ with $\mathsf{level}(i) > a$, the nodes in J for a subset $S_{i,J}$ of the (a, b, γ)-ABTSD scheme can be at most γ levels below $\mathsf{level}(i)$. When $\gamma = a$, the node i will be the root of the a-tree for which nodes in J are leaf nodes. As the value of γ is increased, more and more subsets get added to the collection $\mathcal{S}$. For fixed values of the parameters a, b and γ, denote by $\mathcal{S}_B$ (resp. $\mathcal{S}_C$) the collection of subsets in the (a, b)-ABTSD (resp. (a, b, γ)-ABTSD) scheme. Note that $\mathcal{S}_B \subseteq \mathcal{S}_C$ and $\mathcal{S}_B \setminus \mathcal{S}_C$ consists of subsets $S_{i,J}$ where (i, J) is b-allowed and the difference between $\mathsf{level}(i)$ and the level of nodes in J is more than γ.

One consequence is that the expected header length of the (a, b, γ)-ABTSD scheme will in general be more than that of the (a, b)-ABTSD scheme. At the same time, the user storage required for the (a, b, γ)-ABTSD scheme will in general be lower than that of the (a, b)-ABTSD scheme. For $b = 2^a - 1$, when $\gamma = \ell_0$, the $(a, 2^a - 1, \ell_0)$-ABTSD scheme is the same as the a-ABTSD scheme. When $\gamma < \ell_0$, the user storage of the $(a, 2^a - 1, \gamma)$-ABTSD scheme will be lesser than the a-ABTSD scheme and the expected header length will in general be more than the a-ABTSD scheme.

The cover generation algorithm of the (a, b, γ)-ABTSD is obtained by altering step Step 2)-b)-ii) of Algorithm $\mathcal{C}$ in the following manner. This change is in addition to the change in step Step 2)-b)-iii)-A mentioned for the (a, b)-ABTSD scheme mentioned in Section 5.

Case when $d = 1$ and j_1 is intermediate:
C3:= (There is a $j' \in \mathsf{SDnodes}[j_1]$ such that ($\mathsf{level}(i) - \mathsf{level}(j') > \gamma$)).
If condition C3 is true, add $S_{j_1,\mathsf{SDnodes}[j_1]}$ to $\mathcal{S}_c$ and subsequently mark i as covered and set $\mathsf{SDnodes}[i] = \{i\}$. Else, mark i as intermediate and copy $\mathsf{SDnodes}[j_1]$ to $\mathsf{SDnodes}[i]$.
The key assignment algorithm is also an easy simplification of the key assignment algorithm for the a-ABTSD scheme and we skip the details.

To determine user storage, consider a user u at a leaf node of $\mathcal{T}^0$. In the (a, b)-ABTSD (or the a-ABTSD) scheme, for an ancestor i, a user u gets seeds from $\mathsf{level}(i) - a + 1$ a-trees rooted on the path between i and u. Hence, we needed to consider $1 + 2 + \ldots + (\ell_0 - a + 1)$ a-trees from which keys are assigned to u. For the (a, b, γ)-ABTSD scheme, we need to consider at most $(\gamma - a + 1)$ a-trees for each ancestor i of u. These a-trees have leaf nodes that are at a distance at most γ from an ancestor i of u on the path between u and i. Corresponding to each such a-tree, the user u gets $\xi_{a,b}$ seeds. Counting the seeds corresponding to the NNL-SD subsets, the total user storage is $1 + \ell_0(\ell_0 + 1) + \xi_{a,b}(\ell_0 - a + 1)(\gamma - a + 1)/2$.

Note that the additional number of seeds (over and above those corresponding to the NNL-SD seeds) provided to a user in the (a, b, γ)-ABTSD scheme is $\xi_{a,b}(\ell_0 - a + 1)(\gamma - a + 1)/2$. In contrast, for the (a, b)-ABTSD scheme, the additional number of seeds is $\xi_{a,b}(\ell_0 - a + 1)(\ell_0 - a + 2)/2$. The linear versus the quadratic dependence on ℓ_0 is a consequence of the fact that the number of subsets in $\mathcal{A}$-$\mathcal{S}$ for the (a, b, γ)-ABTSD scheme is restricted by the parameter γ as compared to that in the (a, b)-ABTSD scheme. The consequence on the header length is that the expected header length of the (a, b)-ABTSD scheme will in general be noticeably lower than that of the (a, b, γ)-ABTSD scheme.

Varying choices of a, b and γ give rise to a total of $O(n \log n)$ possible schemes having different storage/header-length trade-off points which lie between the NNL-SD scheme and the power set scheme. For a fixed a, as b or γ is increased one at a time or simultaneously, the number of subsets in $\mathcal{S}$ increases. As a result, the (expected) header length is reduced while the storage requirement increases.

6 Experimental Studies

As we have already noted, the header length is never more than that of the NNL-SD scheme. This, however, does not indicate what will happen on average. In this section, we report on this aspect and also compare the average header length and user storage as a varies.

In order to compute the expected header length, one may consider a situation where r users out of n are randomly revoked without replacement. Then for every non-leaf node i in $\mathcal{T}^0$, one can associate a binary valued random variable X_i which takes the value 1 if a subset of the form $S_{i,j}$ or $S_{i,J}$ is generated and takes the value 0 otherwise. The header length is then $\sum X_i$ and by linearity of expectation, the expected header length is $\sum \Pr[X_i = 1]$.

We have considered the possibility of obtaining an algorithm to compute $\Pr[X_i = 1]$. This, however, becomes too complicated to be useful. Instead, we chose a simulation based approach to get a fair idea of the expected header length. First, we fix the parameter a for the scheme. For given values of n and r, we generate random revocation patterns using Floyd's Algorithm [BF87]. For each such revocation pattern, the cover generation algorithm finds the exact cover and hence we get the header length. The number of iterations is chosen so that the average value of the header length stabilizes. It turns out that 100 iterations are sufficient.

Table 1 shows that for different values of r, the expected header length of the 1-ABTSD scheme (the complete tree version of the NNL-SD scheme) is always more than that of the a-ABTSD scheme with $a > 1$. In fact, as a increases, there is a steep fall in the expected header length for fixed n and r. As an example, we see that for $n = 10^7$ and $r = 0.4n$, the expected header length due to the NNL-SD scheme is 2.29 times that of the a-ABTSD scheme with $a = 3$.

We compare the performance of the a-ABTSD scheme by varying the parameter a. Table 1 shows how the mean header length for a given value of a (MHL_a) varies with n and r. We observe the following:

1. For a fixed n, as the parameter a is increased, the user storage increases.
2. For fixed n and a, the ratio MHL_a/r decreases steadily as r increases. This behavior is true for all $a \geq 1$ (including the NNL-SD scheme).
3. For fixed n and r, as a increases, the ratio MHL_a/r decreases steadily. This holds for any value of r.
4. For fixed a and r/n, the value of MHL_a/r is approximately the same for all values of n. Hence, these properties hold good for the full-tree versions (with $n = 2^{\ell_0}$) of the scheme too.

For certain values of r/n, the ratio MHL_a/r is shown in Table 2. This behavior is further depicted by plotting the values of Table 2 in Figure 5.

The results of the refinements of the (a, b, γ)-ABTSD scheme are shown in the following plots (data for these plots are provided in the corresponding tables as indicated with each figure). Figure 6 shows how the header length is affected by varying $\gamma \in \{a, \ldots, \ell_0\}$ for a fixed $a \in \{3, 4, 5\}$ and $b = 2^a - 1$. It is clear that the expected header length practically stabilises after a point and only the user storage keeps increasing. Figure 7 shows how the header length is affected by varying $b \in \{2, \ldots, 2^a - 1\}$ for a fixed $a \in \{3, 4, 5\}$ and $\gamma = \ell_0$. Increasing the value of b improves the header length while the user storage increases. Figure 8 shows how the header length is affected by varying $a \in \{2, \ldots, 8\}$ for $b \in \{2, 3\}$ and $\gamma = \ell_0$.

7 Comparison to Previous Works

There are three parameters of a BE scheme to be considered, namely, average header size, user storage and the decryption time (required by a user). Schemes in the literature provide a variety of trade-offs between these schemes. Here we discuss how the new schemes compare with the literature.

Table 1: User storage and mean header lengths in the complete a-ABTSD scheme for values of a between 1 and 4. For a fixed n, we report MHL_a/r for three different choices of r namely, $r = (0.1n, 0.2n, 0.4n)$.

n	a	$\mathsf{us}_a(n)$	MHL_a/r	n	a	$\mathsf{us}_a(n)$	MHL_a/r
10^3	1	55	$(1.11, 0.97, 0.71)$	10^4	1	105	$(1.11, 0.97, 0.71)$
	2	145	$(0.96, 0.78, 0.53)$		2	287	$(0.96, 0.78, 0.53)$
	3	1279	$(0.75, 0.53, 0.31)$		3	2757	$(0.75, 0.53, 0.31)$
	4	115247	$(0.52, 0.31, 0.16)$		4	271629	$(0.52, 0.30, 0.16)$
10^5	1	153	$(1.11, 0.97, 0.71)$	10^6	1	210	$(1.11, 0.97, 0.71)$
	2	425	$(0.96, 0.78, 0.53)$		2	590	$(0.96, 0.78, 0.53)$
	3	4233	$(0.75, 0.53, 0.31)$		3	6024	$(0.75, 0.53, 0.31)$
	4	432123	$(0.52, 0.30, 0.16)$		4	629652	$(0.52, 0.30, 0.16)$
10^7	1	300	$(1.11, 0.97, 0.71)$	10^8	1	378	$(1.11, 0.97, 0.71)$
	2	852	$(0.96, 0.78, 0.53)$		2	1080	$(0.96, 0.78, 0.53)$
	3	8902	$(0.75, 0.53, 0.31)$		3	11428	$(0.75, 0.53, 0.31)$
	4	950634	$(0.52, 0.30, 0.16)$		4	1234578	$(0.52, 0.30, 0.16)$

Table 2: Values of the ratio MHL_a/r (for any n) corresponding to the varying ratio r/n for each a. Note that as the value of a increases, the scheme performs better in terms of communication overhead as compared to a lesser value of a.

a \ r/n	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
1	(1.23,	1.18,	1.11,	0.97,	0.84,	0.71,	0.58,	0.46,	0.33,	0.22,	0.11,	0.00)
2	(1.20,	1.08,	0.96,	0.78,	0.64,	0.53,	0.44,	0.35,	0.27,	0.18,	0.10,	0.00)
3	(1.15,	0.93,	0.75,	0.53,	0.39,	0.31,	0.25,	0.20,	0.17,	0.13,	0.08,	0.00)
4	(1.07,	0.73,	0.52,	0.30,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)

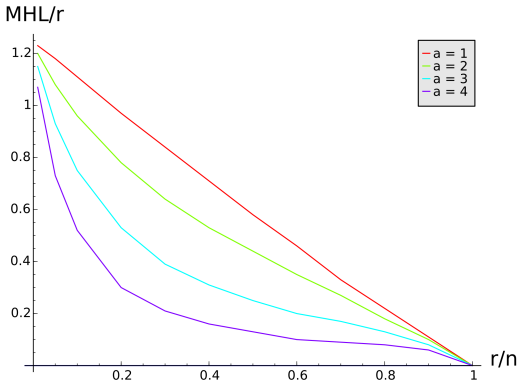


Figure 5: Plot showing how MHL_a/r varies with r/n.

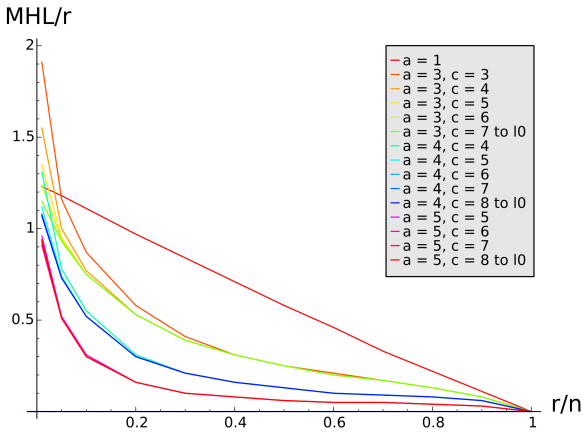
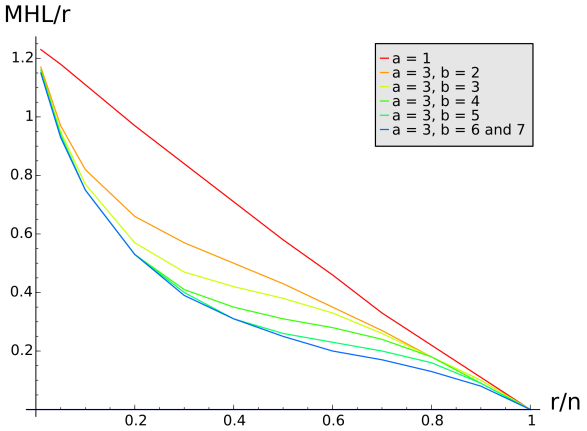


Figure 6: Variation of $\mathsf{MHL}_{(a,b,\gamma)}/r$ with r/n (data from Table 4 of the Supplementray Material) for $a \in \{3,4,5\}$ (independent of n) where $b = 2^a - 1$ and varying $\gamma = c \in \{a, \ldots, \ell_0\}$.

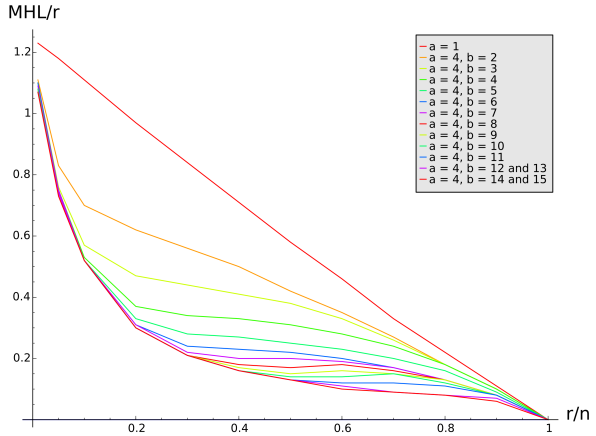
The original NNL-SD scheme as well as the LSD scheme [HS02] require $O(\log n)$ computation time. The goal of the LSD scheme was to modify the NNL-SD scheme so as to reduce the storage requirement at the cost of increasing the header size. This trade-off is in the opposite direction to the trade-off considered in this paper which reduces the header size of the NNL-SD scheme at the cost of increasing the user storage. So, the applicability scenarios of the LSD scheme and the current schemes are different.

The modifications of the NNL-SD scheme provided in [WYL14, WYT13] also requires $O(\log n)$ decryption time. In these works, a scheme called BDSD_λ was introduced where in addition to the NNL-SD subsets, keys were assigned to subsets $S_{i,\{j_1,j_2\}}$ where j_1 and j_2 are two descendants of i on the two child subtrees of i and at the same distance $\lambda + 1$ from i. The parameter λ varied between 1 and $\log n$. As the value of λ is increased, the number of additional subsets increases. This results in (expected) header lengths which are smaller than the NNL-SD scheme.

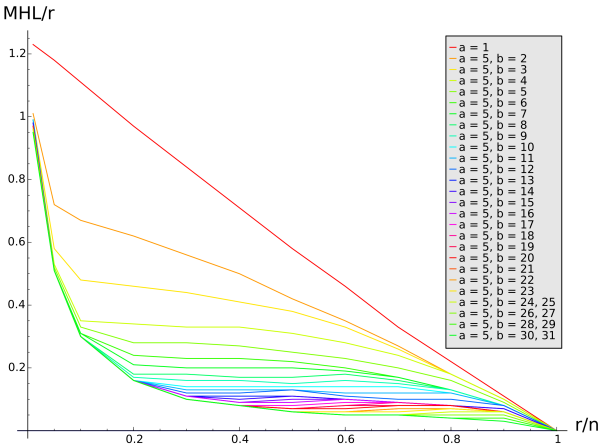
The (a, b, c)-ABTSD scheme with $b = 2$ and $a = c = \lambda$ contains all subsets of the BDSD_λ scheme. Additionally, the $(\lambda, 2, \lambda)$-ABTSD scheme also contains subsets $S_{i,\{j_1,j_2\}}$ where j_1 and j_2 may belong to the same subtree of i. The additional subsets with keys result in the (expected) header length of the $(\lambda, 2, \lambda)$-ABTSD scheme to be better than the BDSD_λ scheme. The variation of the expected header length in the $(\lambda, 2, \lambda)$-ABTSD scheme with the values of $a = \gamma = \lambda$ is shown in Figure 9. The user storage for the BDSD_λ scheme is $O(2^\lambda \log n)$ while that of the $(\lambda, 2, \lambda)$-ABTSD scheme is $O(\xi_{\lambda,2} \log n)$, where in general $\xi_{\lambda,2}$ will be greater than 2^λ. So, the



(a) Plot for $a = 3$ varying $b \in \{2, \ldots, 7\}$ (data from Table 5 of the Supplementray Material).



(b) Plot for $a = 4$ varying $b \in \{2, \ldots, 15\}$ (data from Table 6 of the Supplementray Material).



(c) Plot for $a = 5$ varying $b \in \{2, \ldots, 31\}$ (data from Table 7 of the Supplementray Material).

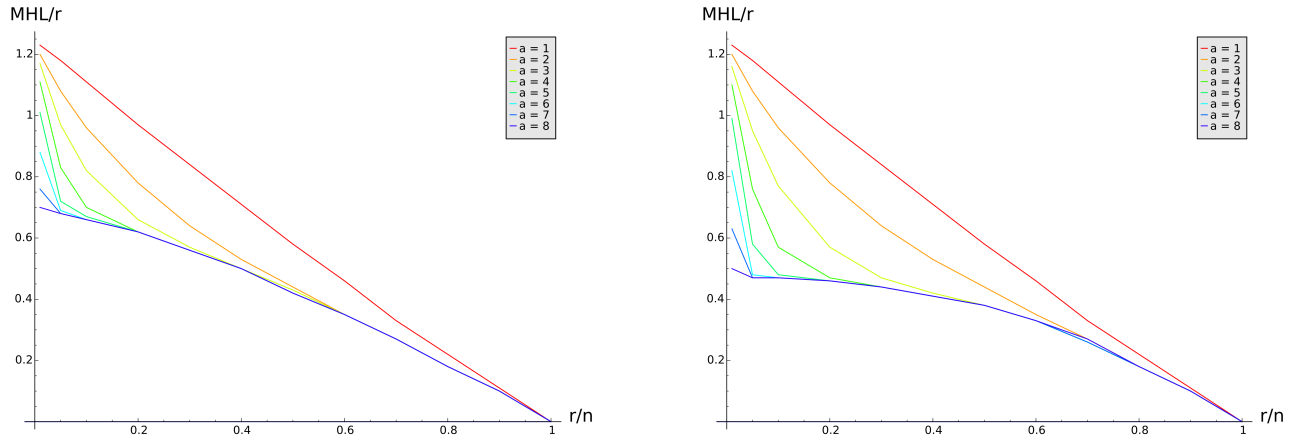
Figure 7: Variation of $\mathsf{MHL}_{(a,b,\gamma)}/r$ with r/n for $a = 4, 5$ (independent of n) and $\gamma = \ell_0$.

$(\lambda, 2, \lambda)$-ABTSD scheme and the BDSD_λ scheme provide different trade-offs.

More generally, the (a, b, c)-ABTSD scheme provides a much larger variety of storage/header-length trade-off points by appropriately varying the parameters a, b and c.

The interesting SSD scheme [GST04] modifies the NNL-SD scheme to reduce the user storage to $O(\log n)$ while ensuring that the header size remains the same as the NNL-SD scheme. The trade-off is that the decryption time increases to $O(n)$. The technique used in [GST04] is interesting and it may be possible to combine this technique with the techniques in the current work. This may lead to schemes which have $O(\log n)$ user storage but lower header sizes. Exploring this option is a possible future work.

The works [JHC+05, CJKY08] describe a scheme which shows several complicated trade-offs between the three basic parameters of a BE scheme. It is difficult to make a direct comparison between this scheme and the present one. Instead, we consider the trade-off given towards the end of Section V of [CJKY08] for $n = 10^8$: the decryption time is about 100 evaluations of a PRG; the user storage is 7633 m-bit strings; and the header size is given in several cases depending upon the value of r. For the 2-ABTSD scheme, the decryption time is about 27 applications of a hash function (following Section 2.1.2, this is also about 27 applications of the underlying PRG); user storage is 1080 m-bit strings; and the expected header size is comparable. Compared to



(a) Plot for $b = 2$ varying $a \in \{1, \ldots, 8\}$ (data from Table 8 of the Supplementray Material).

(b) Plot for $b = 3$ varying $a \in \{1, \ldots, 8\}$ (data from Table 9 of the Supplementray Material).

Figure 8: Variation of $\mathsf{MHL}_{(a,b,\gamma)}/r$ with r/n for $\gamma = \ell_0$ (independent of n).

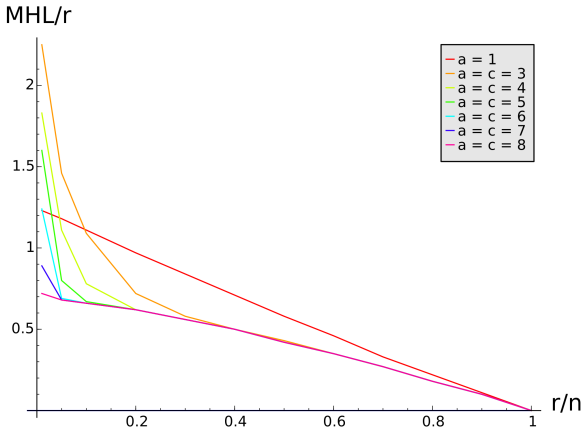


Figure 9: Variation of $\mathsf{MHL}_{(a,b,\gamma)}/r$ with r/n (data from Table 10 of the Supplementray Material) for $a \in \{3, \ldots, 8\}$ (independent of n) where $b = 2$ and $\gamma = a$.

the 3-ABTSD scheme, the decryption time remains same, the user storage goes up to 11428 but, the expected header size comes down. The key issue is that the decryption time of the new schemes is lower and then there is a wide range of trade-offs between the user storage and the header size.

In [HLL05], a generic transformation for BE schemes was proposed to reduce the storage requirement at the cost of increased header length. Two instantiations were described. The first scheme required $O(\log n)$ storage and $O(\log n)$ decryption time, but, $O(r \log n/(\log \log n))$ header size. Taking $n = 10^8$, this shows the header size to be about $5r$ which is more than that of the NNL-SD scheme. The second scheme achieves a different trade-off. Again, taking $n = 10^8$, the decryption time is about 100 evaluations, the storage requirement is about a few thousands m-bit strings and the header size is at most $10^6 + r/2$ which for small r is not good.

8 Conclusion

Broadcast encryption is applied in paid services like cable TV, online broadcasting services (audio, video, gaming and document sharing), content protection in optical discs, etc. for implementing digital rights manage-

ment [DRM]. Starting from the famous NNL-SD scheme [NNL01], several schemes have been proposed in the literature with varying trade-offs.

For many applications, the decryption time required by a user should be small as otherwise there will be a noticeable lag in performance. The least known decryption time is $O(\log n)$ where n is the number of users. The new schemes also achieve this decryption time. Restricted to schemes with $O(\log n)$ decryption time, the schemes proposed in this work provide new trade-offs for reducing the average header size at the cost of increasing the user storage. For many applications, reducing the communication overhead is significantly more important than an increase in the user storage. Such applications will benefit from the trade-offs attainable by the new schemes proposed in this work.

Acknowledgment

We would like to thank the anonymous reviewers for their comments and suggestions which has helped to improve the paper.

References

[AAC] AACS. Advanced Access Content System.

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

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

[BF87] Jon Bentley and Bob Floyd. Programming pearls: a sample of brilliance. *Commun. ACM*, 30(9):754–757, September 1987.

[BS13] Sanjay Bhattacherjee and Palash Sarkar. Complete tree subset difference broadcast encryption scheme and its analysis. *Des. Codes Cryptography*, 66(1-3):335–362, 2013.

[BS14] Sanjay Bhattacherjee and Palash Sarkar. Concrete analysis and trade-offs for the (complete tree) layered subset difference broadcast encryption scheme. *IEEE Trans. Computers*, 63(7):1709–1722, 2014.

[BS15] Sanjay Bhattacherjee and Palash Sarkar. Tree Based Symmetric Key Broadcast Encryption. *J. Discrete Algorithms*, 34:78–107, 2015.

[CJKY08] Jung Hee Cheon, Nam-Su Jho, Myung-Hwan Kim, and Eun Sun Yoo. Skipping, cascade, and combined chain schemes for broadcast encryption. *Information Theory, IEEE Transactions on*, 54(11):5155–5171, Nov 2008.

[DRM] DRM. Digital Rights Management.

[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.

[FKTS08] K. Fukushima, S. Kiyomoto, T. Tanaka, and K. Sakurai. Ternary subset difference method and its quantitative analysis. In *The 9th International Workshop on Information Security Applications (WISA2008)*, volume 5379, pages 225–239. LNCS, 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.

[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.

[HLL05] Jung Yeon Hwang, Dong Hoon Lee, and Jongin Lim. Generic transformation for scalable broadcast encryption schemes. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 276–292. Springer, 2005.

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

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

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

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

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

[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.

[WYL14] Shyh-Yih Wang, Wu-Chuan Yang, and Ying-Jen Lin. Balanced double subset difference broadcast encryption scheme. *Security and Communication Networks*, pages n/a–n/a, 2014.

[WYT13] Shyh-Yih Wang, Wu-Chuan Yang, and Show-Shiow Tzeng. A symmetric-key broadcast encryption scheme with shorter transmissions. In *Next-Generation Electronics (ISNE), 2013 IEEE International Symposium on*, pages 275–278, Feb 2013.

Sanjay Bhattacherjee received his Bachelor of Information Technology Engineering degree in the year 2005 from Jadavpur University, Kolkata and Master of Technology in Computer Science in the year 2009 from the Indian Statistical Institute, Kolkata. He has submitted his Ph.D thesis at the Indian Statistical Institute and has joined the AriC team at ENS-Lyon as a post-doctoral researcher in February 2015. His research interests include cryptology and combinatorics.

Palash Sarkar received his Bachelor of Electronics and Telecommunication Engineering degree in the year 1991 from Jadavpur University, Kolkata and Master of Technology in Computer Science in the year 1993 from the Indian Statistical Institute, Kolkata. He completed his Ph.D from the Indian Statistical Institute in 1999. Since June 2005 he has been a professor at the Indian Statistical Institute. His research interests include cryptology, discrete mathematics and computer science.

Supplementary Material

Reducing User Storage

A user u is provided with the set I_u as secret information. This set is the union of two disjoint sets $I_u^{(1)}$ and $I_u^{(2)}$ where $|I_u^{(1)}| = 1 + \ell_0(\ell_0+1)/2$ and $|I_u^{(2)}| = (2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1)/2$. So the user storage is $|I_u| = 1 + \ell_0(\ell_0+1)/2 + (2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1)/2$ where $k = 2^a$ (see (9)). For a given ℓ_0, the quantity $|I_u^{(1)}| = 1 + \ell_0(\ell_0+1)/2$ is fixed and does not change with the value of a. As the value of a increases, the component $|I_u^{(2)}| = (2^{k-1} - 2k + a + 1)(\ell_0 - a + 2)(\ell_0 - a + 1)/2$ increases. The main increase is due to the exponential factor 2^{k-1} which is actually doubly exponential in a. Here we describe a technique to somewhat mitigate this increase. For small concrete values of a, the decrease in user storage is quite significant.

Recall that the information provided in $I_u^{(2)}$ is used by u to generate keys for the subsets in $\mathcal{A}$-$\mathcal{S}_u$. For a specified value of a, the new key generation method will provide a user u with a different set, to be denoted $\Pi_u^{(2)}(a)$, which will enable u to generate keys for the subsets in $\mathcal{A}$-$\mathcal{S}_u$.

It is to be noted that the technique for decreasing user storage described in this section does not change the definition of the collection $\mathcal{S}$ of subsets to which keys are assigned in the a-ABTSD scheme. Hence, the cover generation algorithm remains the same. Only the method of assigning seeds to nodes and keys to SD subsets is altered.

Suppose the number of users is n. Then as discussed earlier, the user storage is not the same for all users. Denote by $\mathsf{us}_a(n)$ the maximum user storage with n users, i.e., $\mathsf{us}_a(n) = \max_u |I_u|$. For $2^{\ell_0 - 1} < n \leq 2^{\ell_0}$, $\mathsf{us}_a(n) = \mathsf{us}_a(2^{\ell_0})$.

The Basic Idea

Consider a subset $S_{i,J}$ for an allowed pair (i, J). Let j be the a-pivot of J. Then J is a non-simple subset of the set of leaf nodes of $\mathcal{A}_a^j$, i.e., $J \in \mathcal{J}_{ns}(\mathcal{A}_a^j)$. The key $K_{i,J}$ is assigned to $S_{i,J}$ using the hash function H as $K_{i,J} = H[\mathcal{A}_a^j](J, L)$ where j is the a-pivot of J and $L = L_{i,j}$ (5). Let u be a user and consider the set $I_u^{(2)}$. The key $K_{i,J}$ is in $I_u^{(2)}$ if the following condition holds: the a-pivot j of J is an ancestor of u and the ancestor v of u at $\mathsf{level}(J)$ is not in J.

Let $\mathcal{T}$ be a full binary tree of height a having $k = 2^a$ leaf nodes. Any subset J of the leaf nodes of $\mathcal{T}$ can be encoded by a k-bit string $\mathsf{str}(J)$ where the i-th bit from the left of $\mathsf{str}(J)$ is 1 if and only if the i-th leaf node of $\mathcal{T}$ is in J. By extension of this notation, $\mathsf{str}(\mathcal{J}_{ns}(\mathcal{T}))$ denotes the set of k-bit strings encoding the non-simple subsets of $\mathcal{T}$. Define

$$H : \mathsf{str}(\mathcal{J}_{ns}(\mathcal{T})) \times \{0,1\}^m \rightarrow \{0,1\}^m. \tag{13}$$

For $\sigma \in \mathsf{str}(\mathcal{J}_{ns}(\mathcal{T}))$ and $L \in \{0,1\}^m$ define $L_\sigma = H(\sigma, L)$. If w is a leaf node of $\mathcal{T}$, define $\mathsf{keys}[L, \mathcal{T}](w)$ to be the set of all L_σ such that the w-th bit of σ is 0.

Let i be an internal node of $\mathcal{T}^0$ and j be a node of $\mathcal{T}^i$. Let v be a leaf node of the a-tree $\mathcal{A}_a^j$. The seed $L_{i,j}$ is the derived seed from L_i which is assigned to the node j. Let w a leaf node of $\mathcal{A}_a^j$. The keys in $\mathsf{keys}[L_{i,j}, \mathcal{A}_a^j](w)$ are to be made available to users in $\mathcal{T}^w$. This is captured by the following definition.

Using the definition of H in (13), the key $K_{i,J}$ for the subset $S_{i,J}$ is defined to be

$$K_{i,J} \quad = \quad H(\mathsf{str}(J), L_{i,j}) \tag{14}$$

where as before, j is the a-pivot of J. Suppose u is a user. Then the set $I_u^{(2)}$ is the following.

$$I_u^{(2)} \quad = \quad \bigcup_i \bigcup_j \mathsf{keys}[L_{i,j}, \mathcal{A}_a^j](v) \tag{15}$$

where i is an ancestor of u; j is node on the path from u to i and $\mathsf{level}(j) \geq a$; v is the ancestor of u at level $\mathsf{level}(j) - a$.

Our basic idea of reducing key storage is that instead of directly providing $\mathsf{keys}[L_{i,j}, \mathcal{A}_a^j](v)$ we provide sufficient information for the keys in this set to be computed. This is achieved by defining the function H in a different manner. Note that the function H can itself be defined with respect to a full binary tree $\mathcal{T}$ of height a and without reference to the tree $\mathcal{T}^0$. Once H is defined, the definition of $K_{i,J}$ follows and the set $\mathsf{keys}[L_{i,j}, \mathcal{A}_a^j](v)$ is also obtained from the definition of $\mathsf{keys}[L, \mathcal{T}](w)$.

In the rest of this section, we show how to define suitable H. In the next subsection, we describe this method for the special case of $a = 2$ and in the subsequent subsection we consider the case of general a.

The Case $a = 2$

For $a = 2$, $k = 2^a = 4$. For $a = 2$, the factor $2^{k-1} - 2k + a + 1 = 3$ and so from (9) the maximum number of seeds to be stored by a user is

$$1 + \ell_0(\ell_0 + 1)/2 + 3\ell_0(\ell_0 - 1)/2. \tag{16}$$

We show how to reduce the factor 3 to 2 by suitably defining the function H.

Let $\mathcal{T}$ be a full binary tree of height a. Then the simple subsets of $\mathcal{T}$ are encoded by the 6 strings $0001, 0010, 0100, 1000, 0011, 1100$ and the non-simple subsets of $\mathcal{T}$ are encoded by the 8 strings $0101, 0110, 0111, 1001, 1010$. So, given an m-bit string L and a string σ encoding a non-simple subset of $\mathcal{T}$, we have to define $L_\sigma = H(\sigma, L)$. Let the leaf nodes of $\mathcal{T}$ from the left be $\theta_0, \ldots, \theta_3$. Then

$\mathsf{keys}[L, \mathcal{T}](\theta_0) = \{L_{0101}, L_{0110}, L_{0111}\}$;
$\mathsf{keys}[L, \mathcal{T}](\theta_1) = \{L_{1001}, L_{1010}, L_{1011}\}$;
$\mathsf{keys}[L, \mathcal{T}](\theta_2) = \{L_{0101}, L_{1001}, L_{1101}\}$;
$\mathsf{keys}[L, \mathcal{T}](\theta_3) = \{L_{0110}, L_{1010}, L_{1110}\}$.

Each of these sets contains 3 m-bit strings which gives the factor 3 in (16). Since L and $\mathcal{T}$ will be clear from the context we will drop them from the notation. We show how to define H such that any of the sets $\mathsf{keys}(\theta_0), \ldots, \mathsf{keys}(\theta_3)$ can be obtained from 2 m-bit strings.

We define a new tree T_4. This tree has no relation to the tree $\mathcal{T}^0$. It is solely used to define the function H. The tree T_4 is defined as follows. The root node has four children nodes numbered $0, 1, 2, 3$. The child node numbered i has two children numbered $(i, 0)$ and $(i, 1)$. The structure is shown in Figure 10.

Define, two hash functions $F_1 : \{0,1,2,3\} \times \{0,1\}^m \to \{0,1\}^m$ and $F_2 : \{0,1\} \times \{0,1\}^m \to \{0,1\}^m$. These hash functions are chosen by the broadcast center and made available to the users in the system.

Given an m-bit seed L, define

$$\left.\begin{array}{rcll} \widehat{L}_i & = & F_1(i, L) & \text{for } i = 0,1,2,3; \\ \widehat{L}_{i,b} & = & F_2(b, \widehat{L}_i) & \\ & & = F_2(b, F_1(i, L)) & \text{for } i = 0,1,2,3 \\ & & & \text{and } b = 0,1. \end{array}\right\} \tag{17}$$

Define

$L_{1110} = \widehat{L}_{1,0}$, $L_{1010} = \widehat{L}_{1,1}$, $L_{1101} = \widehat{L}_{2,0}$,
$L_{0101} = \widehat{L}_{2,1}$, $L_{1011} = \widehat{L}_{3,0}$, $L_{1001} = \widehat{L}_{3,1}$,
$L_{0111} = \widehat{L}_{4,0}$, $L_{0110} = \widehat{L}_{4,1}$.

Then each of the sets $\mathsf{keys}(\theta_0), \ldots, \mathsf{keys}(\theta_3)$ can be obtained from 2 m-bit seeds as indicated below.

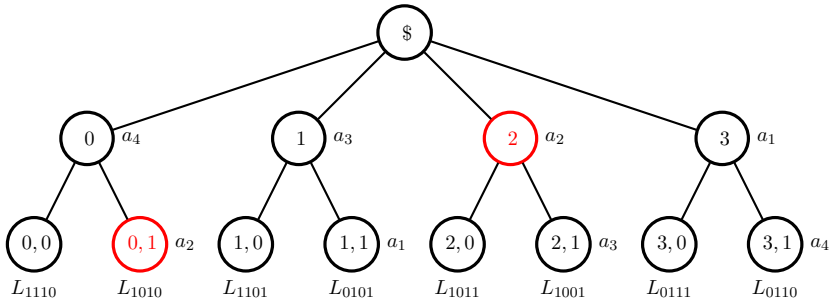


Figure 10: The structure of T_4 for $a = 2$.

$\mathsf{keys}(\theta_0)$: $\widehat{L}_3$ and $\widehat{L}_{1,1}$;
$\mathsf{keys}(\theta_1)$: $\widehat{L}_2$ and $\widehat{L}_{0,1}$;
$\mathsf{keys}(\theta_2)$: $\widehat{L}_1$ and $\widehat{L}_{2,1}$;
$\mathsf{keys}(\theta_3)$: $\widehat{L}_0$ and $\widehat{L}_{3,1}$.

It is easy to verify that the above information is sufficient to obtain any set $\mathsf{keys}(\theta_i)$. For example, the users under the node $4j + 3$ in $\mathcal{T}^0$ will be able to get the seeds $\{L_{0101}, L_{0110}, L_{0111}\}$.

Fix a user u and an ancestor i of u at level ℓ. For every node j which is an ancestor of u at levels between 2 and ℓ, the set $\mathrm{II}_u^{(2)}(2)$ contains two m-bit seeds. Since ℓ can vary from 2 to ℓ_0, we have

$$|\mathrm{II}_u^{(2)}(2)| \quad = \quad 2 \times \frac{\ell_0(\ell_0 - 1)}{2} = \ell_0(\ell_0 - 1). \tag{18}$$

Based on this we obtain the following improvement to (16).

$$\mathsf{us}_2(2^{\ell_0}) \quad = \quad 1 + \ell_0(\ell_0 + 1)/2 + \ell_0(\ell_0 - 1). \tag{19}$$

General Case

The technique for $a = 2$ is somewhat specific since in this case the number of non-simple subsets of an a-tree turns out to be 8 which is a power of 2. More generally, Lemma 1 shows that the number of non-simple subsets of an a-tree is $2^k - 2k$ where $k = 2^a$. The expression $2^k - 2k$ will not be a power of 2 for $a > 2$. For this case, we directly use the technique from [BS15] which dealt with the same problem in a different context. We explain this below.

The k-ary Tree Subset Difference Scheme

The underlying structure of the NNL-SD scheme is the binary tree $\mathcal{T}^0$. The work [BS15] generalizes the idea to work with k-ary trees for any $k \geq 2$. So, suppose that $\mathcal{T}^0$ is a k-ary tree. Then each internal node has k children. Let i be an internal node of $\mathcal{T}^0$ and J be a non-empty subset of nodes having a common parent j. Let $S_{i,J}$ denote the leaf nodes of the graph formed by taking away from $\mathcal{T}^0$ the subtrees whose root nodes are in J. The collection $\mathcal{S}$ for the k-ary tree scheme consists of all such subsets $S_{i,J}$.

Key assignment in the k-ary tree scheme is done as follows. Each node is assigned a seed L_i and a hash function is iteratively used to define the seed $L_{i,j}$ for any node j in the subtree rooted at i. Given $L_{i,j}$ and the subset J of children nodes of j, a key $L_{i,J}$ is defined. In [BS15] this is first defined directly and then later it is shown how to define this in a different manner so that the user storage reduces.

Coming back to the a-ABTSD scheme, we note the similarity between the subsets and the key assignment procedure of the two schemes. The relevant difference is that in the k-ary tree scheme the subset J is a non-empty

Table 3: Effect of reduction of user storage. In the second row the entry for $a = 2$ is from (18) and the entries for $a = 3$ and $a = 4$ are from (20).

storage	$a = 2$	$a = 3$	$a = 4$
$2\|I_u^{(2)}\|/(\ell_0 - a + 2)(\ell_0 - a + 1)$ from (8)	3	116	32741
$2\|\mathrm{II}_u^{(2)}(a)\|/(\ell_0 - a + 2)(\ell_0 - a + 1)$	1	36	4116

subset of the children nodes of j, whereas in the a-ABTSD scheme, the subset J is a non-simple subset of the leaf nodes of the a-tree rooted at j. For both cases, the key to $S_{i,J}$ is assigned from the seed $L_{i,j}$. So, in both cases the problem is given an m-bit seed L and the subset J, how to define the key based on L and J?

Using Cyclotomic Cosets to Reduce User Storage

A solution to this problem has been given in [BS15] which uses the notion of cyclotomic cosets. We do not provide the solution here and instead refer the reader to [BS15] for details. Our main observation is that the solution provided in [BS15] also works in the present case. The difference is that the method of [BS15] assigns keys to all non-empty subsets of the children nodes of j, whereas in the present case, we only need to assign keys to all non-simple subsets of the leaf nodes of the a-tree rooted at j. This difference, however, is not significant. We simply ignore the keys that are assigned to the simple subsets.

On the other hand, it is also possible to actually modify the key assignment procedure in [BS15] so that keys are only assigned to non-simple subsets. We have carried this out for $a = 3$ and $k = 2^a = 8$. The work required us to examine the $2^k - 1 = 256$ non-empty subsets and eliminate the keys assigned to $2k - 1 = 15$ simple subsets. These details are quite tedious and so we do not report them. Directly using the key assignment procedure from [BS15] in the present context shows that $\mathrm{II}_u^{(2)}(a)$ for a user u consists of $(\chi_k - 2)(\ell_0 - a + 2)(\ell_0 - a + 1)/2$ m-bit keys where χ_k is the number of cyclotomic cosets of k-bit strings, i.e., for $a > 2$,

$$\mathrm{II}_u^{(2)}(a) = \frac{(\chi_{2^a} - 2) \times (\ell_0 - a + 2)(\ell_0 - a + 1)}{2}. \tag{20}$$

So, for $a > 2$,

$$\begin{aligned} \mathsf{us}_a(2^{\ell_0}) &= 1 + \frac{\ell_0(\ell_0 + 1)}{2} \\ &\quad + \frac{(\chi_{2^a} - 2) \times (\ell_0 - a + 2)(\ell_0 - a + 1)}{2}. \end{aligned} \tag{21}$$

For the case of $a = 2$ and $k = 4$, $\chi_4 = 6$. Hence, from (21) $\mathsf{us}_2(2^{\ell_0})$ would be $1 + \ell_0(\ell_0 + 1)/2 + 2\ell_0(\ell_0 - 1)$. Previously, however, we have seen that $\mathsf{us}_2(2^{\ell_0}) = 1 + \ell_0(\ell_0 + 1)/2 + \ell_0(\ell_0 - 1)$. So, for the case of $a = 2$, directly using the solution from [BS15] is sub-optimal. This is one of the reasons why we considered the case of $a = 2$ as a special case.

For small value of a the reduction that is achieved is shown in Table 3. It is clear that the reduction achieved is significant in practical terms.

Application to the (a, b)-aABTSD Scheme. We consider whether the user storage of the (a, b)-aABTSD scheme can be further reduced using the cyclotomic coset idea. For $a = 2$, the parameter b can be either 2 or 3; for $b = 3$, we obtain the 2-ABTSD scheme and this has already been discussed. There is a possibility that the

$(2,2)$-ABTSD scheme may lead to user storage lower than that of the 2-aABTSD scheme. This, however, does not turn out to be true and the user storage for the $(2,2)$-ABTSD scheme is the same as that of the 2-ABTSD scheme. So, for $a = 2$, there is no motivation for considering the $(2,2)$-ABTSD scheme. On the other hand, for $a \geq 3$ and $b < 2^a - 1$, it is possible to obtain schemes with reduced user storage using the cyclotomic coset idea. Closed formulas for these cases are complicated to obtain and so we do not present them.

Correctness of the Cover Finding Algorithm

The cover generation algorithm outputs sets of the type $S_{j,\mathsf{SDnodes}[j]}$. To show the correctness of the algorithm we need to argue two things.

1. Each subset produced by Algorithm $\mathcal{C}$ is in $\mathcal{S}$.
2. The subsets that are produced form a partition of the set of privileged users.

Lemma 3. *If Algorithm $\mathcal{C}$ produces a subset $S_{i,J}$, then every element of J has been marked* covered.

Proof. J is of the form $\mathsf{SDnodes}[j]$ for some node j. Further, all nodes in $\mathsf{SDnodes}[j]$ are marked covered. This can be seen from the manner in which the $\mathsf{SDnodes}[j]$ is built up. Nodes enter $\mathsf{SDnodes}[j]$ either in Step 1 or in Step 2(b)(i) and in both cases they are marked covered; the set $\mathsf{SDnodes}[j]$ grows in Step 2(b)(iii)(B) through the union of two other sets of the same type and hence the property of having only covered nodes is preserved. □

Lemma 4. *If a subset $S_{i,J}$ is produced by Algorithm $\mathcal{C}$, then J is a reduced set.*

Proof. All nodes in J are marked covered. Let if possible j_1 and j_2 be siblings in J and i is their parent. Then both j_1 and j_2 are marked covered. When the node i is considered in Step 2(b), then d is 2 and Step 2(b)(i)(B) is executed which results in $\mathsf{SDnodes}[i]$ being set to $\{i\}$ and j_1, j_2 do not enter any $\mathsf{SDnode}[i]$. So, they cannot be members of any J such that $S_{i,J}$ is produced by Algorithm $\mathcal{C}$ at a later point of time. □

Lemma 5. *For any set $\mathsf{SDnodes}[j]$, if $i_1, i_2 \in \mathsf{SDnodes}[j]$, then $\mathsf{level}(i_1) - \mathsf{level}(i_2) < a$. Further, all nodes of $\mathsf{SDnodes}[j]$ belong to some a-tree.*

Proof. Let $J = \mathsf{SDnodes}[j]$. If J is a singleton set, then this is clearly true; if J contains more than one element, then J must have been formed by the merger of two SDnodes set in Step 2(b)(iii)(B). Such merger can take place only if the maximum of the differences in the levels of the nodes in the resulting set is less than a.

For the last statement, again it is easy to see this if J is a singleton set. On the other hand, if J has been formed by merger (one or more times), then each such merger is a union of the SDnodes of two siblings. Consequently, this corresponds to a moving-up operation within the same a-tree. □

Lemma 6. *Any subset produced by Algorithm $\mathcal{C}$ is in the collection $\mathcal{S}$.*

Proof. Suppose $S_{j,\mathsf{SDnodes}[j]}$ is produced. Then all the nodes in $J = \mathsf{SDnodes}[j]$ are in the subtree rooted at j. By Lemma 5, the nodes in J are in some a-tree and by the previous statement, the root of this a-tree is also in $\mathcal{T}^j$. So, $S_{j,J}$ is in $\mathcal{S}$. □

Lemma 7. *If u is a leaf node corresponding to a revoked user, then Algorithm $\mathcal{C}$ visits all ancestors of u.*

Proof. Whenever a node i is processed by Algorithm $\mathcal{C}$, its parent is added to $\mathcal{L}$. Further, every node in $\mathcal{L}$ is processed before the algorithm terminates. Since the initial list $\mathcal{L}$ contains the node u, every ancestor of u is processed by Algorithm $\mathcal{C}$. □

Lemma 8. *Any privileged (i.e., non-revoked) user is in one of the subsets produced by Algorithm $\mathcal{C}$.*

Proof. Let v be a privileged user. Since there is at least one revoked user, there is a minimal subtree $\mathcal{T}^i$ of $\mathcal{T}^0$ which contains both v and some revoked user u. Let j_1 and j_2 be the two children of i and suppose v is a leaf node of $\mathcal{T}^{j_2}$. By the minimality of $\mathcal{T}^i$, it follows that u is necessarily in $\mathcal{T}^{j_1}$ and further all leaf nodes of $\mathcal{T}^{j_2}$ are privileged.

Since i is an ancestor of the revoked node u, by the previous lemma, Algorithm $\mathcal{C}$ will process both nodes i_1 and i. The node i is added to $\mathcal{L}$ when node i_1 is processed. Since all nodes in $\mathcal{T}^{i_2}$ are privileged, node i_2 does not enter $\mathcal{L}$. So, i has exactly one child in $\mathcal{L}$ and either by Step 2(b)(i)(A) or by Step 2(b)(ii), i is marked $\mathsf{intermediate}$ and $\mathsf{SDnodes}[i]$ is set to either $\{j_1\}$ or to $\mathsf{SDnodes}[j_1]$. In both cases, v is in $S_{i,\mathsf{SDnodes}[i]}$. From this point onwards, Algorithm $\mathcal{C}$ ensures the following. If i' is an ancestor of i, then either the set $S_{i',\mathsf{SDnodes}[i']}$ is produced, or, $S_{i',\mathsf{SDnodes}[i']}$ contains v. Since, the second case cannot continue indefinitely, at some point of time, Algorithm $\mathcal{C}$ will produce a set $S_{i',\mathsf{SDnodes}[i']}$ for some ancestor i' of i and so v will be in this subset. □

From Lemmas 7 and 8, we get the following result on the correctness of Algorithm $\mathcal{C}$.

Theorem 9. *Algorithm $\mathcal{C}$ produces a sub-collection of subsets of $\mathcal{S}$ which form a partition of the set of privileged users.*

The complexity of Algorithm $\mathcal{C}$ is given by the following theorem.

Theorem 10. *Algorithm $\mathcal{C}$ requires $O(r \log n)$ time where r is the number of revoked nodes.*

Proof. As proved in Lemma 7, the algorithm processes every ancestor of any revoked node. There are $O(\log n)$ such ancestors and so the total time taken by the algorithm is proportional to $r \log n$. □

It has already been remarked that for $a = 1$, the a-ABTSD scheme collapses to the NNL-SD scheme. Theorem 11 shows that for $a > 1$ and any revocation pattern, the header length of the a-ABTSD scheme is never more than that of the NNL-SD scheme. The proof has been provided in the supplementary material.

Theorem 11. *For a given $\mathcal{R}$ (revocation pattern) the header length due to the NNL-SD scheme is at least as large as that of the a-ABTSD scheme.*

Proof. For a given value of a, let $\mathcal{J}_a$ be the collection of all nodes j in $\mathcal{T}^0$ such that $S_{j,\mathsf{SDnodes}[j]} \in \mathcal{S}_c$. Let us consider a node i in $\mathcal{T}^0$ that have both children $\{j_1, j_2\}$ in $\mathcal{L}$ and at least one of them is marked as $\mathsf{intermediate}$. When $a = 1$, for every $\mathsf{intermediate}$ child j of i, there is a $j' \in \mathsf{SDnodes}[j]$ such that $\ell_j - \ell_{j'} \geq 1$. Hence, $S_{j,\mathsf{SDnodes}[j]} \in \mathcal{S}_c$ and hence $j \in \mathcal{J}_{a=1}$. For $a > 1$, if for some $j \in \{j_1, j_2\}$, there is a $j' \in \mathsf{SDnodes}[j]$ such that $\ell_j - \ell_{j'} \geq a$, only then all intermediate children of i generate SD subsets. Otherwise, i is marked as $\mathsf{intermediate}$ and $\mathsf{SDnodes}[j]$ is included in $\mathsf{SDnodes}[i]$ and is carried upwards. Hence, $\mathcal{J}_{a=1} \subseteq \mathcal{J}_{a>1}$. Thus, the header length due to a revocation pattern for the a-ABTSD scheme will be at most that of the NNL-SD scheme. □

It follows from Theorem 11 above that the worst case header length for the a-ABTSD scheme will be less than or equal to that of the NNL-SD scheme. From [NNL01] we know that for a given r, the worst case header length of the NNL-SD scheme is $2r - 1$. Hence we get the following theorem.

Theorem 12. *For a given r in the a-ABTSD scheme, the maximum header length that can be achieved for any n, is $2r - 1$.*

This upper bound is tight, as we discuss next. We consider the a-ABTSD scheme with $a = 2$ for $n = 32$ users in Figure 11 where $\mathcal{R} = \{31, 39\}$. The subset cover for this revocation pattern is $\mathcal{S}_c = \{S_{3,\{31\}}, S_{4,\{39\}}, S_{0,\{1\}}\}$. Hence, the header length is $2|\mathcal{R}| - 1 = 3$. A similar example can be constructed to show the tightness of this upper bound for any general value of a with larger values of n. The subtrees rooted at nodes $3, 4, 5$ and 6 in Figure 11 where $a = 2$, are of height $a + 1 = 3$ each. For any general a, these subtrees should be full subtrees of height $a + 1$ each. It is to be noted that the tree $\mathcal{T}^0$ in such a case will be of height $a + 3$ and the total number

of users will be 2^{a+3}. There will be two revoked users, one in each of the subtrees rooted at nodes 3 and 4. The subset cover will have three subsets. Two of these subsets will be rooted at nodes 3 and 4. The third subset will be $S_{0,\{3,4\}} = S_{0,\{1\}}$. Hence, the upper bound given by Theorem 12 is tight for any $a \geq 1$.

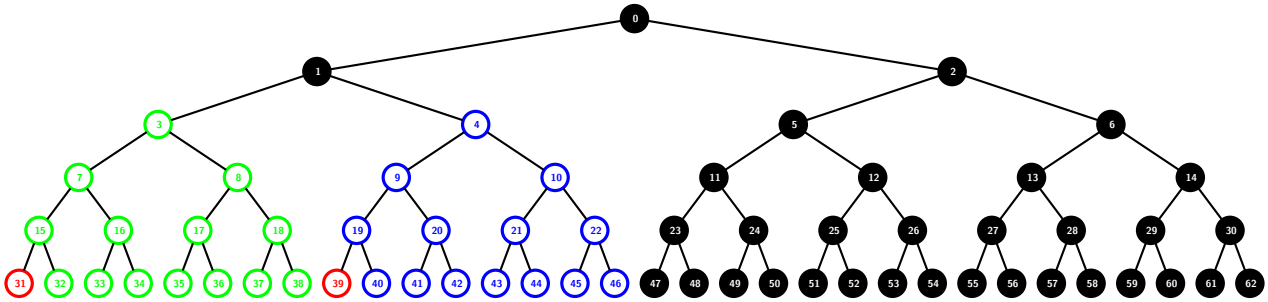


Figure 11: Example to show that the upper bound $2r - 1$ of the header length in the a-ABTSD scheme with $a = 2$ is tight. The subset cover for $\mathcal{R} = \{31, 39\}$ in the binary tree $\mathcal{T}^0$ with $n = 32$ users contains the subsets $S_{3,\{31\}}$, $S_{4,\{39\}}$ and $S_{0,\{1\}}$.

Other Issues

In this section, we comment on three issues. The issues are important and the reason for discussing them in the supplementary material is that previously known techniques apply in a straightforward manner to the present construction.

Accommodating Arbitrary Number Of Users

The NNL-SD [NNL01] scheme assumes the number of users n to be a power of two. The a-ABTSD scheme retains this assumption and hence assumes an underlying full binary tree. In practice this may be restrictive. We extend the a-ABTSD scheme for arbitrary number of users by assuming a *complete* binary tree instead of full. A complete binary tree with $2^{\ell_0 - 1} < n \leq 2^{\ell_0}$ leaves is formed by adding child nodes to the leaf nodes of a full tree with $2^{\ell_0 - 1}$ leaf nodes, starting from the left. These newly added leaves are said to be at level 0. The old leaves are at level 1. The newly constructed complete tree has n leaves, some of which are filled from the left of level 0 and the others (if $2^{\ell_0 - 1} < n < 2^{\ell_0}$) are on the right at level 1.

Since the underlying tree $\mathcal{T}^0$ is a complete tree (that may not be full) and hence an a-tree may also be a non-full complete binary tree. Thus, an a-tree $\mathcal{A}_a^i$ is a complete tree rooted at node i in $\mathcal{T}^0$ and is of height a. Let us call the path joining the root node and the right-most internal node at level 1 to be the *dividing path*. Any subtree of $\mathcal{T}^0$ rooted at a node other than the dividing path, is full. Hence, only the a-tree rooted at the node on the dividing path at level a may be non-full. The subsets that are included in the collection $\mathcal{S}$ are formed as before. A subset $S_{i,J} \in \mathcal{S}$ is such that all nodes in J are within a single (possibly non-full but complete) a-tree.

The user storage requirement of the a-ABTSD scheme assuming $n = 2^{\ell_0}$ is given by (9) where ℓ_0 is the height of the underlying tree. Let us denote this storage requirement as $\mathsf{us}_a(2^{\ell_0})$. Then the user storage of the scheme assuming the complete tree structure will be at least $\mathsf{us}_a(2^{\ell_0 - 1})$ and at most $\mathsf{us}_a(2^{\ell_0})$, depending on where a user is placed in the tree with respect to the dividing path. All users are attached to some node on the dividing path. Users that are to the left (respectively right) of the dividing path and are attached to it at nodes on or above level a, receive $\mathsf{us}_a(2^{\ell_0})$ seeds (respectively $\mathsf{us}_a(2^{\ell_0 - 1})$ seeds). For the users that are attached to the dividing path at a level less than a, the number of seeds can be easily calculated from the number of users attached to the dividing path at those levels.

The cover generation algorithm for the complete tree version of the scheme would have an additional pre-processing step for the leaf nodes at level 0. First, all the revoked leaf nodes at level 0 are inserted into the list $\mathcal{L}$ in left-to-right order. These nodes are processed one after another as in the cover generation algorithm. The parent of each leaf in $\mathcal{L}$ gets appended to it and their respective data structures are appropriately updated. Once all revoked leaves at level 0 have been processed, all their parents at level 1 are in the list. The remaining revoked leaf nodes that are at level 1 in $\mathcal{T}^0$, are then appended to $\mathcal{L}$. Then onwards, the cover generation algorithm proceeds exactly as it did for full trees. The worst-case header length remains $2r-1$ for the complete tree version of the scheme. We have implemented this algorithm and results are reported later.

Full Resilience

A user obtains secret information I_u which allows it to obtain a set of keys. Let us denote this set as $\mathcal{K}_u$. It is to be noted that under certain reasonable cryptographic assumptions on the hash functions G, F_1 and F_2, user u does not obtain any information about keys that are not in $\mathcal{K}_u$. Further, if $\mathcal{K}$ is a set of keys and $\mathcal{U}_{\mathcal{K}}$ is the set of all users such that $\mathcal{K} \cap \mathcal{K}_u = \emptyset$, then $\cup_{u \in \mathcal{U}} \mathcal{K}_u$ does not provide any information about $\mathcal{K}$ (again under reasonable cryptographic assumptions on G, F_1 and F_2). This can be argued formally along the lines of the argument provided in [NNL01]. We skip the details and only remark that this can be intuitively seen by considering the hash functions to be one-way and the outputs of the hash functions to be independent.

Traitor Tracing

Traitor tracing is an important feature of BE schemes. It is the mechanism to identify leaked user keys from a pirate decoder by treating it as a "black-box". Traitor tracing for the NNL-SD scheme was discussed in details in [NNL01]. They showed that traitor tracing can be done on any scheme that assigns keys to subsets which satisfy the *bifurcation property*. The bifurcation property states that *given any subset that is in the collection $\mathcal{S}$ and hence has been assigned a key, it is possible to partition the set into two (or a constant number of) almost equal subsets from $\mathcal{S}$*. The *bifurcation value* is defined to be the ratio of the size of the largest subset to that of the set itself. For the BE schemes of [HS02, BS13, BS14], the subsets used in these schemes all belong to the collection $\mathcal{S}$ for the NNL-SD scheme with the same number of users. Hence, their respective traitor tracing mechanisms are almost the same as the NNL-SD scheme.

For the a-ABTSD scheme that we have proposed in this work, keys are assigned to subsets that are in general different from those in the NNL-SD scheme. Hence, the traitor tracing for these schemes do not directly follow from the NNL-SD traitor tracing algorithm. However, the subsets of this scheme do follow the bifurcation property. Here we state very briefly how these subsets can be split into roughly equal sized subsets from their respective collection $\mathcal{S}$.

In the a-ABTSD scheme, the subsets in the collection $\mathcal{S}$ are of the forms $S_{i,j}$ or $S_{i,J}$. Any subset of the form $S_{i,j}$ can also be written as $S_{i,J}$ where J is a simple subset of $\mathcal{A}_a^j$. Assume that all subsets in $\mathcal{S}$ are of the form $S_{i,J}$ where J is a non-empty subset of the leaf nodes of $\mathcal{A}_a^j$ for some j in the subtree rooted at i. Subsets where $J = \{j\}$ is a singleton set are split into two as was done in [NNL01]. The node j will be in either of the two subtrees rooted at $2i+1$ or $2i+2$. If j is in $\mathcal{T}^{2i+1}$, the subsets after split will be $S_{2i+1,j}$ and $S_{i,2i+1}$. If j is in $\mathcal{T}^{2i+2}$, the subsets after split will be $S_{2i+2,j}$ and $S_{i,2i+2}$. Hence, the maximum bifurcation value in this case is $2/3$.

For the subsets $S_{i,J}$ where $|J| > 1$, let us consider the a-tree $\mathcal{A}_a^i$ rooted at node i. The a-tree $\mathcal{A}_a^j$ containing the nodes in J is either this same a-tree (when $i = j$) or it is rooted at a descendant j of i. In any case, the subsets formed by the split are as follows. The subtrees rooted at leaves of $\mathcal{A}_a^i$ form a subset each in the split. From each of these 2^a subtrees, all users under nodes in J are excluded. As a result, some of these 2^a subtrees may be completely excluded. When $i = j$, the maximum bifurcation value is $1/(2^a - |J|)$ which in the worst case would be $1/2$. In case j is in the subtree of i, the nodes in J will be contained in at least one (but not all) of the

Table 4: For $a \in \{3,4,5\}$, $b = 2^a - 1$, $\gamma \in \{a, \ldots, \ell_0\}$, the values of the ratio $\mathsf{MHL}_{(a,b,\gamma)}/r$ varying with the ratio r/n. These values are same for all values of $\ell_0 \geq 20$. The user storages are reported for $\ell_0 = 20$.

$a;\gamma$ \ r/n	$\mathsf{us}_a(n)$	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
$3;3$	498	(1.91,	1.15,	0.87,	0.57,	0.41,	**0.31**,	**0.25**,	**0.21**,	**0.17**,	**0.13**,	**0.08**,	0.00)
$3;4$	786	(1.55,	1.00,	0.77,	**0.53**,	**0.39**,	0.31,	0.25,	0.21,	0.17,	0.13,	0.08,	0.00)
$3;5$	1074	(1.35,	0.95,	**0.75**,	0.53,	0.39,	0.31,	0.25,	0.21,	0.17,	0.13,	0.08,	0.00)
$3;6$	1362	(1.18,	0.94,	0.75,	0.53,	0.39,	0.31,	0.25,	0.21,	0.17,	0.13,	0.06,	0.00)
$3;7,\ldots,\ell_0$	≥ 1650	(**1.15**,	**0.93**,	0.75,	0.53,	0.39,	0.31,	0.25,	0.21,	0.17,	0.13,	0.08,	0.00)
$4;4$	35136	(0.00,	0.00,	0.55,	0.31,	**0.21**,	**0.16**,	**0.13**,	**0.10**,	**0.09**,	**0.08**,	**0.06**,	**0.00**)
$4;5$	70063	(0.00,	0.00,	**0.52**,	**0.30**,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)
$4;6$	104989	(0.00,	0.00,	0.52,	0.30,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)
$4;7,\ldots,\ell_0$	≥ 139916	(**0.00**,	**0.00**,	0.52,	0.30,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)

Table 5: For $a = 3$, $b \in \{2, \ldots, 7\}$, $\gamma = \ell_0$, the values of the ratio $\mathsf{MHL}_{(a,b,\gamma)}/r$ varying with the ratio r/n. These values are independednt of n.

b \ r/n	$\mathsf{us}_a(n)$	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
2	2154	(1.17,	0.97,	0.82,	0.66,	0.57,	0.50,	0.43,	0.35,	0.27,	0.18,	0.10,	0.00)
3	3288	(1.16,	0.95,	0.77,	0.57,	0.47,	0.42,	0.38,	0.33,	0.26,	0.18,	0.10,	0.00)
4	4422	(1.16,	**0.94**,	**0.75**,	**0.53**,	0.41,	0.35,	0.31,	0.28,	0.24,	0.18,	0.10,	0.00)
5	5070	(1.15,	0.94,	0.75,	0.53,	**0.40**,	**0.31**,	0.26,	0.23,	0.20,	0.16,	0.09,	0.00)
6, 7	5394	(**1.15**,	0.94,	0.75,	0.53,	0.40,	0.31,	**0.25**,	**0.20**,	**0.17**,	**0.13**,	**0.08**,	0.00)

2^a subtrees under the a-tree $\mathcal{A}_a^i$. The users in the subtrees of J are excluded from the respective subtrees at the leaves of $\mathcal{A}_a^i$. Since j is in the subtree of i, one of the child subtrees of i would not have any node in J. There will be at least 2^{a-1} subtrees at the leaves of $\mathcal{A}_a^i$ that will not have any node in J. As a result, the bifurcation value in this case will be between $1/2^{a-1}$ and $1/2^a$. This goes to show that the bifurcation property also holds for subsets in the a-ABTSD scheme. Hence, traitor tracing mechanisms can be devised for the scheme introduced in this work in a manner similar to the one described in [NNL01].

The number of queries required by the traitor tracing algorithm depends on the bifurcation value. At every step of the traitor tracing algorithm, a subset S of users that contains a traitor is divided into subsets $S_1, \ldots, S_t$ using the bifurcation property as mentioned above. Each subset S_t is tested for containment of a traitor. The ratio $|S_t|/|S|$ is at most the bifurcation value. Lesser the bifurcation value, lesser is the size of the remaining subset from which the traitors have to be traced. The bifurcation value of the NNL-SD scheme is $2/3$. The bifurcation value of the a-ABTSD scheme is at most $2/3$ for $a \geq 2$. Hence, traitor tracing in the a-ABTSD scheme will be at least as efficient as the NNL-SD scheme, if not better on an average.

Experimental Data

Table 6: For $a = 4$, $b \in \{2, \ldots, 15\}$, $\gamma = \ell_0$, the values of the ratio $\mathsf{MHL}_{(a,b,\gamma)}/r$ varying with the ratio r/n. These values are independent of n.

b \ r/n	$\mathsf{us}_a(n)$	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
2	6279	(1.11,	0.83,	0.70,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)
3	19139	(1.10,	0.76,	0.57,	0.47,	0.44,	0.41,	0.38,	0.33,	0.26,	0.18,	0.10,	0.00)
4	52519	(1.10,	0.75,	0.53,	0.37,	0.34,	0.33,	0.31,	0.28,	0.24,	0.18,	0.10,	0.00)
5	116243	(1.10,	0.75,	0.52,	0.33,	0.28,	0.27,	0.25,	0.23,	0.20,	0.16,	0.09,	0.00)
6	212191	(1.10,	0.75,	0.52,	0.31,	0.24,	0.23,	0.22,	0.20,	0.17,	0.13,	0.08,	0.00)
7	324757	(1.09,	0.75,	0.52,	0.31,	0.22,	0.20,	0.20,	0.19,	0.17,	0.13,	0.08,	0.00)
8	429953	(1.08,	0.74,	0.52,	0.30,	0.21,	0.18,	0.17,	0.18,	0.16,	0.13,	0.08,	0.00)
9	511162	(1.08,	0.74,	0.52,	0.30,	0.21,	0.17,	0.15,	0.16,	0.15,	0.13,	0.08,	0.00)
10	560581	(1.08,	0.74,	0.52,	0.30,	0.21,	0.16,	0.14,	0.14,	0.15,	0.12,	0.08,	0.00)
11	583412	(1.07,	0.74,	0.52,	0.30,	0.21,	0.16,	0.13,	0.12,	0.12,	0.11,	0.08,	0.00)
12	591359	(1.07,	0.74,	0.52,	0.30,	0.21,	0.16,	0.13,	0.11,	0.09,	0.08,	0.07,	0.00)
13	593671	(1.07,	0.74,	0.52,	0.30,	0.21,	0.16,	0.13,	0.11,	0.09,	0.08,	0.07,	0.00)
14	593960	(1.07,	0.73,	0.52,	0.30,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)
15	593960	(1.07,	0.73,	0.52,	0.30,	0.21,	0.16,	0.13,	0.10,	0.09,	0.08,	0.06,	0.00)

Table 7: For $a = 5$, $b \in \{2, \ldots, 15\}$, $\gamma = \ell_0$, the values of the ratio $\mathsf{MHL}_{(a,b,\gamma)}/r$ varying with the ratio r/n. These values are independent of n.

b \ r/n	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
2	(1.01,	0.72,	0.67,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)
3	(0.99,	0.58,	0.48,	0.46,	0.44,	0.41,	0.38,	0.33,	0.26,	0.18,	0.10,	0.00)
4	(0.99,	0.53,	0.35,	0.34,	0.33,	0.33,	0.31,	0.28,	0.24,	0.18,	0.10,	0.00)
5	(0.99,	0.52,	0.33,	0.28,	0.28,	0.27,	0.25,	0.23,	0.20,	0.16,	0.09,	0.00)
6	(0.99,	0.51,	0.31,	0.24,	0.23,	0.23,	0.22,	0.20,	0.17,	0.13,	0.08,	0.00)
7	(0.99,	0.51,	0.31,	0.21,	0.20,	0.20,	0.20,	0.19,	0.17,	0.13,	0.08,	0.00)
8	(0.99,	0.51,	0.30,	0.18,	0.18,	0.17,	0.17,	0.18,	0.16,	0.13,	0.08,	0.00)
9	(0.99,	0.51,	0.30,	0.17,	0.16,	0.16,	0.15,	0.16,	0.15,	0.13,	0.08,	0.00)
10	(0.99,	0.51,	0.30,	0.16,	0.14,	0.14,	0.14,	0.14,	0.14,	0.12,	0.08,	0.00)
11	(0.99,	0.51,	0.30,	0.16,	0.13,	0.13,	0.13,	0.12,	0.12,	0.12,	0.08,	0.00)
12	(0.98,	0.51,	0.30,	0.16,	0.12,	0.12,	0.13,	0.11,	0.10,	0.10,	0.08,	0.00)
13	(0.98,	0.51,	0.30,	0.16,	0.11,	0.11,	0.11,	0.10,	0.09,	0.08,	0.07,	0.00)
14	(0.98,	0.51,	0.30,	0.16,	0.11,	0.10,	0.11,	0.10,	0.09,	0.08,	0.07,	0.00)
15	(0.97,	0.51,	0.30,	0.16,	0.11,	0.09,	0.10,	0.10,	0.09,	0.08,	0.06,	0.00)
16	(0.97,	0.51,	0.30,	0.16,	0.11,	0.09,	0.09,	0.10,	0.09,	0.08,	0.06,	0.00)
17	(0.97,	0.51,	0.30,	0.16,	0.10,	0.08,	0.08,	0.09,	0.09,	0.08,	0.06,	0.00)
18	(0.97,	0.51,	0.30,	0.16,	0.10,	0.08,	0.07,	0.08,	0.09,	0.08,	0.06,	0.00)
19	(0.97,	0.51,	0.30,	0.16,	0.10,	0.08,	0.07,	0.08,	0.08,	0.08,	0.06,	0.00)
20	(0.97,	0.51,	0.30,	0.16,	0.10,	0.08,	0.07,	0.07,	0.08,	0.08,	0.06,	0.00)
21	(0.97,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.06,	0.07,	0.07,	0.06,	0.00)
22	(0.96,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.06,	0.07,	0.07,	0.06,	0.00)
23	(0.96,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.06,	0.06,	0.07,	0.06,	0.00)
24	(0.96,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.05,	0.05,	0.06,	0.06,	0.00)
25	(0.96,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.05,	0.05,	0.06,	0.06,	0.00)
26	(0.95,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.05,	0.05,	0.05,	0.05,	0.00)
27	(0.95,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.05,	0.05,	0.05,	0.05,	0.00)
28	(0.95,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.05,	0.05,	0.04,	0.04,	0.00)
29	(0.95,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.05,	0.05,	0.04,	0.04,	0.00)
30	(0.95,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.05,	0.05,	0.04,	0.03,	0.00)
31	(0.95,	0.51,	0.30,	0.16,	0.10,	0.08,	0.06,	0.05,	0.05,	0.04,	0.03,	0.00)

Table 8: For $a \in \{1, \ldots, 6\}$, $b = 2$ for $a > 1$, $\gamma = \ell_0$, the values of the ratio $\mathsf{MHL}_{(a,b,\gamma)}/r$ varying with r/n. These values are independent of n.

a \ r/n	$\mathsf{us}_a(n)$	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
1	210	(1.23,	1.18,	1.11,	0.97,	0.84,	0.71,	0.58,	0.46,	0.33,	0.22,	0.11,	0.00)
2	571	(1.20,	1.08,	0.96,	0.78,	0.64,	0.53,	0.44,	0.35,	0.27,	0.18,	0.10,	0.00)
3	2154	(1.17,	0.97,	0.82,	0.66,	0.57,	0.50,	0.43,	0.35,	0.27,	0.18,	0.10,	0.00)
4	6279	(1.11,	0.83,	0.70,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)
5	19410	(1.01,	0.72,	0.67,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)
6	147997	(0.88,	0.69,	0.66,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)

Table 9: For $a \in \{1, \ldots, 6\}$, $b = 3$ for $a > 1$, $\gamma = \ell_0$, the values of the ratio $\mathsf{MHL}_{(a,b,\gamma)}/r$ varying with r/n. These values are independent of n.

a \ r/n	$\mathsf{us}_a(n)$	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
1	210	(1.23,	1.18,	1.11,	0.97,	0.84,	0.71,	0.58,	0.46,	0.33,	0.22,	0.11,	0.00)
2	751	(1.20,	1.08,	0.96,	0.78,	0.64,	0.53,	0.44,	0.35,	0.27,	0.18,	0.10,	0.00)
3	3288	(1.16,	0.95,	0.77,	0.57,	0.47,	0.42,	0.38,	0.33,	0.26,	0.18,	0.10,	0.00)
4	19139	(1.10,	0.76,	0.57,	0.47,	0.44,	0.41,	0.38,	0.33,	0.26,	0.18,	0.10,	0.00)
5	132690	(0.99,	0.58,	0.48,	0.46,	0.44,	0.41,	0.38,	0.33,	0.26,	0.18,	0.10,	0.00)
6	2201839	(0.82,	0.48,	0.47,	0.46,	0.44,	0.41,	0.38,	0.33,	0.26,	0.18,	0.10,	0.00)

Table 10: For $a = \gamma \in \{3, \ldots, 8\}$ and $b = 2$, the values of the ratio $\mathsf{MHL}_{(a,b,\gamma)}/r$ varying with r/n. These values are independent of n.

a \ r/n	$\mathsf{us}_a(n)$	(0.01,	0.05,	0.10,	0.20,	0.30,	0.40,	0.50,	0.60,	0.70,	0.80,	0.90,	1.00)
3	318	(2.25,	1.46,	1.09,	0.72,	0.58,	0.50,	0.43,	0.35,	0.27,	0.18,	0.10,	0.00)
4	567	(1.83,	1.11,	0.78,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)
5	1410	(1.60,	0.80,	0.67,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)
6	4395	(1.24,	0.69,	0.66,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)
7	15204	(0.89,	0.68,	0.66,	0.62,	0.56,	0.50,	0.42,	0.35,	0.27,	0.18,	0.10,	0.00)