Study of Broadcast Encryption and New Approaches Towards It

M.Tech. Dissertation work of:
Sanjay Bhattacherjee
Rooll No: MTC0707

supervised by:
Professor Palash Sarkar
Applied Statistics Unit

INDIAN STATISTICAL INSTITUTE
203, B. T. ROAD,
KOLKATA, INDIA - 700108.

Abstract

This report is a study of Broadcast Encryption (BE) schemes primarily concentrating on the widely used *Subset Cover Framework*. We start with the basic framework of Broadcast Encryption, defining the various terminologies that are found in the BE literature. The study of schemes concentrates primarily on the Complete Subtree and the Subset Difference schemes. We also study the more recent work, the Punctured Interval scheme. We summarize the results of the significant schemes at the end of our study.

As our contribution, we propose two new frameworks: the *Hitting Set Framework* and the *Interval Framework*. We describe the Hitting Set Framework in which finding the minimal set of keys for a transmission can be mapped to the bipartite matching problem which has well known polynomial time solutions. Then we describe the Interval Framework and propose two new schemes in this framework. The first scheme, L-DAG gives results similar to the Punctured Interval scheme ($N - 1$ keys per user for N users for the header size of $r + 1$ where there are r revoked users). The other I-L-DAG scheme achieves much better results ($\log N$ keys per user for the header size of r) but at the cost of resilience. We also propose a new tradeoff between keys per user and the header size using our first scheme L-DAG in which, increasing the header size to $r + k - 1$ from r, we can decrease the number of keys by a factor of k making it $2N/k - 1$. We compare our scheme with the Punctured Interval scheme, and suggest some improvements to the latter.

Then we propose an improvisation on the tree-based schemes: the *k-ary tree* scheme based on the subset-cover framework. In this, we combine the ideas used in the Complete Subtree scheme and the Subset Difference scheme to perform the key pre-distribution with the help of k-ary trees where $k \geq 3$. This helps us achieve a reduced number of keys per user while the header size grows to $3r - 2$. As an example, we start with the *Ternary Tree* scheme from which we develop the idea for the more general k-ary tree scheme. Further, in the k-ary tree scheme, for a fixed N, we study the variation in the number of keys stored per user by varrying k. We argue that there will be a k for which the number of keys stored per user will be minimmum.

Preface

This is the report of our work for my M. Tech. dissertation at the Indian Statistical Institute with the able guidance of Professor Palash Sarkar.

Structure of the report

The report is distributed essentially into three parts. The first part introduces Broadcast Encryption and the real-life problems from where this entire area of research arises. In the second part we study the significant works in the area of Broadcast Encryption starting with the work of Amos Fiat and Moni Naor [FN93]. Then we go through the Subset Cover Framework by Dalit Naor, Moni Naor and Jeff Lotspiech [NNL01], and then the improvised LSD scheme by Dani Halevy and Adi Shamir [HS02]. Finally we study the most recent work by Nam-Su Jho, Jung Hee Cheon, Myung-Hwan Kim and Eun Sun Yoo [JHC$^+$05] which is one of the best schemes as per our study. In the third part, we put our contributions that include two new frameworks and a few new schemes. Our L-DAG scheme gives a result similar to the one in [JHC$^+$05]. It also gives new tradeoffs in the number of keys and the header size. Another scheme proposed by us, the k-ary tree scheme is an improvisation on the existing tree-based schemes. It also gives new trade-offs between the number of keys and the header size.

Prerequisites

Basic counting techniques, concept of pseudorandom generators, cryptographic security notions based on computational security and imagination (trivially).

Implementations

I have implemented a program that finds all possible subsets generated by the Subset Difference scheme in [NNL01] accommodating the labeling of the users in a cyclic manner. I have also implemented the subset cover algorithm of our k-ary Tree scheme, that takes as input the set of revoked users and finds the minimal set of subsets (to which keys have been pre-assigned) such that they cover all privileged users.

Acknowledgments

I would like to heartily thank Professor Palash Sarkar who has been thoroughly instrumental in making every bit of this dissertation work possible. He introduced the subject of Broadcast Encryption to me, informed me about online resources and textbooks, motivated me to do a background study and made sure I presented my progress regularly to him. He corrected every mistake I made in the process, and had utmost patience to listen to whatever I wanted to discuss. He motivated me to think in a manner different from what I had read in the literature. This helped me come up with a lot of new ideas that gave me immense satisfaction and confidence to move ahead.

He taught me how to do research. We started off from ideas that were trivial and basic and then slowly discussed and developed on them and finally came up with a scheme that gave results similar to one of the best known schemes till date. I felt his equal involvement (if not more) in the entire thought process. He kept communicating even on non-working days. He pointed out my strengths and also informed me about the areas where I could improve. I have been trying to immitate his disciplined approach and learn from him, the importance of formalism in research. Thank you Palashda.

I would also like to thank Professor Bimal Roy, whose course on Cryptography introduced me to the subject. I would also like to thank Dr. Anupama Panigrahi who taught us preliminary Number Theory.

I would like to thank Mr. Sumit Pandey, who, in my initial days of study for this dissertation, helped me get preliminary notions on Cryptology. I would also like to thank Mr. Rishiraj Bhattacharyya, whom I can find beside me whenever I need him. I would like to thank my friend, Miss Somindu C. R. who helped me at diffrent times with different resourses, technical or otherwise.

I would like to thank my fiancé, Miss Sanchari Mukherjee, who always stood by me and gave a patient hearing to all my wiered ideas. I would like to thank Ma, Mrs. Sabita Bhattacherjee, who has been my biggest support all through my life. And finally, I would like to thank my Baba, Mr. Atirindra Nath Bhattacherjee, who was the first person to tell me that I should be doing research someday.

Contents

Part I

Background

Chapter 1

What is Broadcast Encryption ?

1.1 Necessity is the mother of invention

With the advent of the information era, as the emphasis on the free transfer of information grew, one significant challenge that the world economy has been facing is *information piracy*. Pay-TV subscribers found ways to circumvent the techniques adopted to prevent viewing of channels without payment. Digital media could be easily copied for reuse. Security was sought at all levels including the information processing at the users. Data had to be encrypted so that only legitimate users have access to it.

1.2 The first few steps

Broadcast Encryption frameworks assume a scenario where there is a center and a set of users. The center provides the users with prearranged keys when they join the system. At some point, the center wishes to broadcast a message (e.g. a session key for the subsequent session) to a *dynamic* subset of users in such a way that non-privileged users cannot learn the message.

According to [JHC+05], *Broadcast Encryption* (BE) is a cryptographic method for a center to efficiently broadcast digital contents to a large set of users so that only a subset of users for whom the content is intended (called privileged users) can decrypt the contents. BE has a wide range of applications such as internet or mobile video broadcast, Pay-TV or even digital media like CD or DVD to name a few.

In context of the BE framework, a few simple solutions that are immediate are as follows (mentioned in [FN93]):

First Simple Solution Provide every user its own key and transmit an individually encrypted message to every member of the privileged class. This would require a very long transmission (number of members in the privileged class times the length of the message).

Second Simple Solution Provide every possible subset of users with a key. This would mean giving every user the keys corresponding to every subset it belongs to. The message length would not be affected in this case by the number of privileged users, since there would be a unique key corresponding to the subset of privileged users, and only that key would be used for encryption.

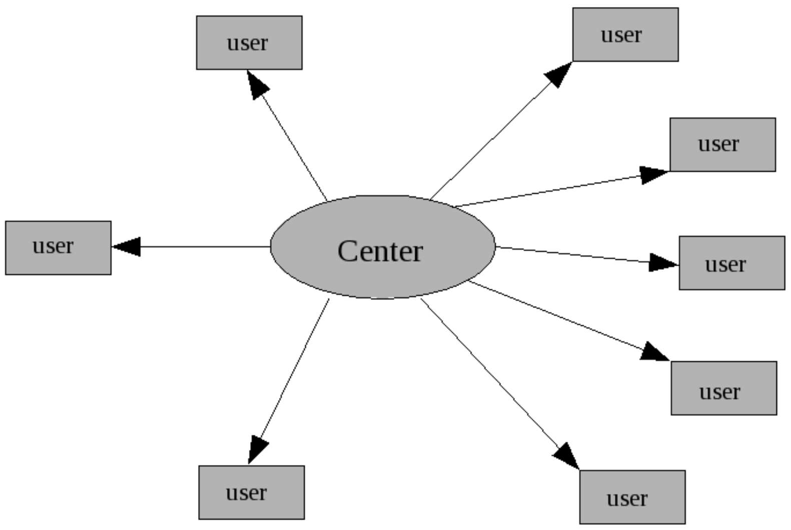


Figure 1.1: BE Framework showing a Center and many Users

So, the transmitted message would have just one encryption of the secret. But the number of keys per user will be very large.

As we will see soon, our interest in the study of BE primarily revolves around optimization of different parameters associated with the different schemes proposed. The above simple solutions are two extremes with respect to the parameters *keys per user* and *transmission overhead*. All the schemes that we have studied in this context, try to optimize these parameters in different directions.

1.3 A few definitions

In BE, the center distributes to each user u, the set K_u of keys called the *user key set* of u. We assume that the user keys are not updated afterward i.e.; the user keys are *stateless*. A *session* is a time interval during which only one encrypted message (digital content) is broadcast. The *session key* (say) SK is the key used to encrypt the contents of a session.

Definition 1.3.1. *A broadcast scheme allocates keys to users so that given a subset* T *of* U*, the center can broadcast messages to all users following which all members of* T *have a common key. This allocation of keys is called* ***key pre-distribution****.*

In order to broadcast a message M, the center encrypts M using the session key SK and broadcasts the encrypted message together with a *header* which contains the encryptions of SK such that it can be retrieved by non-revoked users. In other words, the center broadcasts

$$\langle header; E_{SK}(M)\rangle$$

where E_{SK} is a symmetric encryption algorithm running with the key SK.

Every non-revoked user u computes $F(K_u, header)$ to find SK from the header and then decrypt $E_{SK}(M)$ with SK where F is a predefined algorithm. For any revoked user v, $F(K_v, header)$ should not render SK.

Moreover, there should be no polynomial time algorithm that outputs SK even with all the revoked user keys and the header as input. This is where the idea of *resilience* follows from.

Definition 1.3.2. *A broadcast scheme is called* ***resilient*** *to a set* S *($S \subset U$) if for every subset* T *that does not intersect with* S *($T \cap S = \phi$), any eavesdropper (adversary) that has all secrets associated with members of* S, *cannot obtain any "knowledge" of the secret common to* T.

Definition 1.3.3. *A scheme is called* ***k-resilient*** *if it is resilient to any set $S \subset U$ of size k.*

The transmissions in a BE system are of three types: key pre-distribution (typically done during manufacture of set-top-boxes in a Pay-TV system), secondly, the privileged and the revoked sets are identified (by strings encoding the identification of privileged subsets of users of the system) typically as part of the header for a session and finally, the actual encryption of the broadcast content. The later two are defined as follows:

Definition 1.3.4. *When nothing is known about the privileged subset T, any broadcast scheme requires that an initial set of messages be exchanged between the center and all the users so that they can have initial set of their own individual private keys or data structures agreed upon. This is called* ***Set Identification Transmission****.*

Definition 1.3.5. *For every session being established between the center and the privileged set of users, messages are broadcast by the center to all users to establish a common key for that session using which all transmissions will be encrypted by the center so that only the privileged set of users (having the common key) can decrypt it. Such messages form the* ***Broadcast Encryption Transmission****.*

Part II

Existing Schemes

Chapter 2

Where to start? (Zero Message Schemes)

One can recollect the mention in section 1.2 of the fact that most of the existing literature in BE proposes schemes fitting as solutions to the problem and then analyzes it with respect to the parameters that are of interest to see their applicability in real life at different BE scenarios. If one intends to improve the different schemes, the following "resources" are points of interest of the existing schemes :

- *Message Header Length* for a session.
- *Number of Keys stored at the center.*
- *Number of Keys stored at each user.*
- *Computation Effort* involved in retrieving the common key by the members of the privileged class.

Other than these, the *resilience* of a scheme is also of interest.

We start with the work of Fiat-Naor [FN93]. Zero Message Schemes do not require the center to broadcast any message in order for the member of the privileged class to generate a common key. (Note: It is assumed that the Key Pre-distribution during the Set Identification Transmission for a session has already happened or would take negligible bandwidth.)

2.1 The Basic Scheme

The Basic Scheme allows users to determine a common key for every subset, resilient to any set S of size k.

Definition 2.1.1. *For every set $B \subset S$, $0 \leq |B| \leq k$, define a key K_B and give K_B to every user $x \in U \setminus B$. The common key to the privileged set T is simply the exclusive or of all keys K_B, $B \subset U \setminus T$, $0 \leq |B| \leq k$.*

Clearly, every coalition of $S \leq k$ users will all be missing key K_S and will therefore be unable to compute the common key for any privileged set T such that $S \cap T$ is empty.

Subset Name	Subset	Key Assigned
S_1	$\{u_1\}$	k_1
S_2	$\{u_2\}$	k_2
S_3	$\{u_3\}$	k_3
S_4	$\{u_4\}$	k_4
S_5	$\{u_1, u_2\}$	k_5
S_6	$\{u_1, u_3\}$	k_6
S_7	$\{u_1, u_4\}$	k_7
S_8	$\{u_2, u_3\}$	k_8
S_9	$\{u_2, u_4\}$	k_9
S_{10}	$\{u_3, u_4\}$	k_{10}

Table 2.1: Example key assignment to subsets for 2-resilient Basic scheme for 4 users

User	Keys Assigned
u_1	$\{k_2, k_3, k_4, k_8, k_9, k_{10}\}$
u_2	$\{k_1, k_3, k_4, k_6, k_7, k_{10}\}$
u_3	$\{k_1, k_2, k_4, k_5, k_7, k_9\}$
u_4	$\{k_1, k_2, k_3, k_5, k_6, k_8\}$

Table 2.2: Key pre-distribution to subsets for the above key assignment to subsets

Let us consider an example with four users $U = \{u_1, u_2, u_3, u_4\}$ that have been assigned independent random keys as shown in table 2.1. User u_1 gets the keys for subsets to which it does not belong. Table 2.1 shows the key pre-distributuion. Now, if the only privileged user is u_1, then the key to be used to encrypt the session key SK would be

$$k_2 \oplus k_3 \oplus k_4 \oplus k_8 \oplus k_9 \oplus k_{10}.$$

It can be easily observed that unless all the other three users $\{u_2, u_3, u_4\}$ collide, the key of u_1 cannot be found. This as expected complies with the above argument.

The memory requirement for this scheme is that every user is assigned $\sum_{i=1}^{k} \binom{n-1}{i}$ keys. With this requirement we need make no assumptions whatsoever. We therefore have:

Theorem 2.1.1. *There exists a k-resilient scheme that requires each user to store* $\sum_{i=1}^{k} \binom{n-1}{i}$ *keys and the center need not broadcast any message in order to generate a common key to the privileged class.*

The center needs to store $\sum_{i=1}^{k} \binom{n}{i}$ keys.

2.2 1-Resilient Schemes using Cryptographic Assumptions

Trying to improve the memory requirements of the basic scheme, cryptographic assumptions such as "one-way functions exist" and "extracting prime roots modulo a composite is hard" are useful. These improvements are applicable to any k. However, they are most dramatic for $k = 1$.

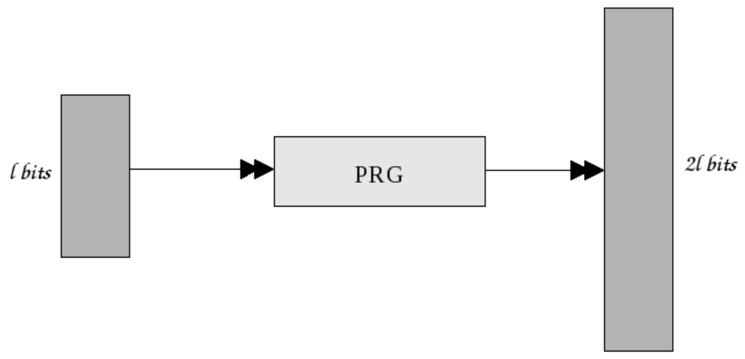


Figure 2.1: Pseudorandom Generator (a black box that doubles a uniform random string)

2.2.1 1-Resilient Scheme based on One-Way Functions

If we consider the 1-resilient version of the basic scheme described above, it requires every user to store $n-1$ different keys. This will be reduced to $\lceil \log n \rceil$ keys per user if the keys are pseudorandomly generated from a common seed. The following is the way this can be done.

Assumption : We assume that one-way functions exist and hence pseudorandom generators exist. Let $G : \{0,1\}^l \rightarrow \{0,1\}^{2l}$ be the pseudorandom generator such that the length of the output of G is twice the length of the input of G as shown in figure 2.1.

Key Distribution : The n users are associated with the leaves of a binary tree on n nodes. The root is labeled with the common seed $s \in \{0,1\}^l$. The other vertices are labeled recursively as follows: *apply the pseudorandom generator G to the root label and taking the left half (first l bits) of $G(s)$ to be the label of the left subtree while the right half bits (last l bits) of $G(s)$ to be the label of the right subtree.*

By the basic scheme, every user x should get all the keys except the one associated with the singleton set $B = \{x\}$. To let a user x have all the other keys, the path from the root to the leaf corresponding to that user is removed which results in a forest of $\lceil \log n \rceil$ trees. Upon this, the user x is provided with the labels associated with the roots of these trees. Hence, whenever needed, user x can compute the labels associated with the leaf nodes (except that of $K_{\{x\}}$).

Resilience : Given the labels of the roots of all the trees in the above forest, the label of $K_{\{x\}}$ is still pseudorandom for user x. If two or more users belonging to the two subtrees of the root come together, they can know the keys corresponding to all subsets in the system (they will have the keys for the roots of the two subtrees of the root of the tree from which keys for all nodes of the tree can be derived).

Theorem 2.2.1. *If one-way functions exist, then there exists a 1-resilient scheme that requires each user to store $\log n$ keys and the center need not broadcast any message in order to generate a common key to the privileged class.*

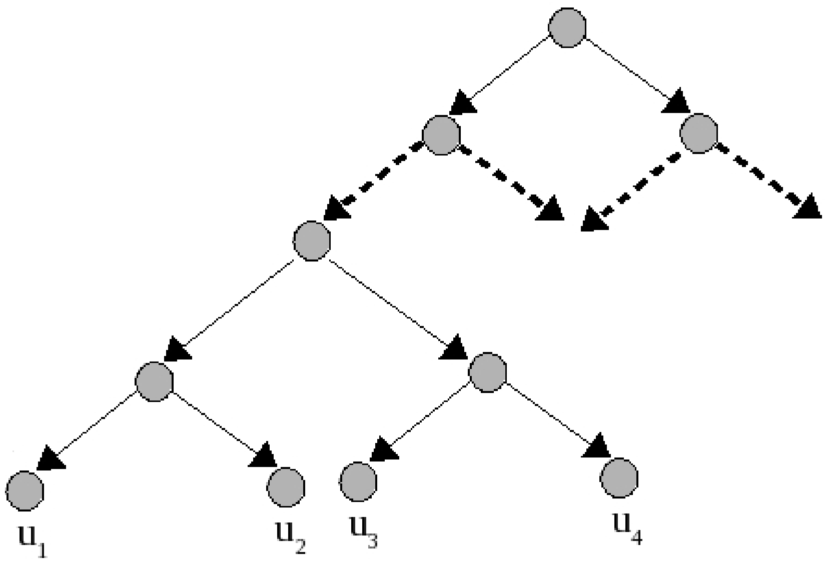


Figure 2.2: Complete Binary Tree to whose leaves the users are associated. (Only a portion of the bottom of the tree has been shown. The dotted lines indicate there are more nodes below them but could not be shown due to space constraints.) This tree having N leaves has $N-1$ internal nodes.

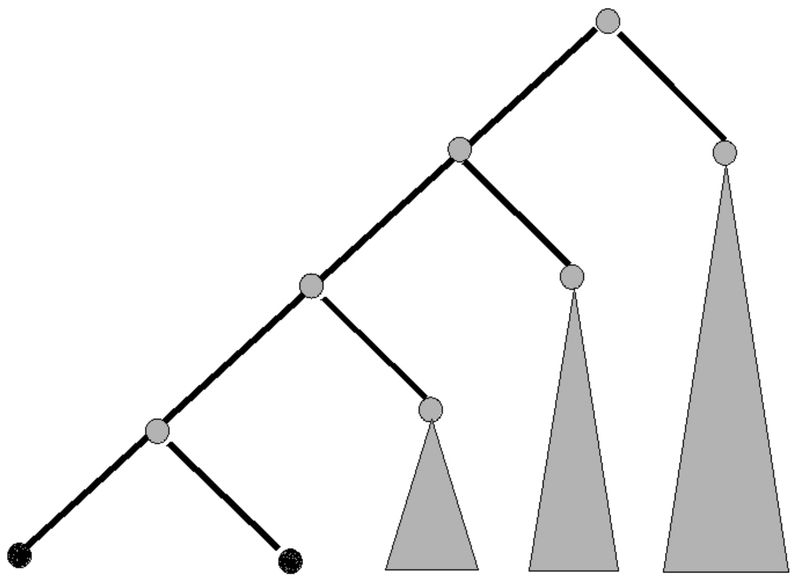


Figure 2.3: The subtrees falling off from the path between the root and the leftmost leaf (user) in a Complete Binary Tree

2.2.2 A 1-Resilient Scheme based on Computational Number Theoretic Assumptions

A specific number theoretic scheme, cryptographically equivalent to the problem of root extraction modulo a composite, can further reduce the memory requirements for $1 - \textit{resilient}$ schemes. This scheme is cryptographically equivalent to the RSA scheme and motivated by the Diffie-Hellman key exchange mechanism, and the original Shamir cryptographically secure pseudorandom sequence.

Key Distribution The center chooses a hard-to-factor composite $N = P.Q$ where P and Q are primes. It also chooses a secret value g of high index. User i is assigned key $g_i = g^{p_i}$, where p_i, p_j are relatively prime for all $i, j \in U$. All users know what user index refers to what p_i. A common key for a privileged subset of users T is taken as the value $g_T = g^{p_T} \mod N$ where $p_T = \prod_{i \in T} p_i$.

Every user $i \in T$ can compute g_T by evaluating

$$g_i^{\prod_{j \in T - \{i\}} p_j} \mod N$$

Resilience Suppose that for some $T \subset U$ and some $j \notin T$, user j could compute the common key for T. Then j could also compute g. If this is assumed to be hard, then any user outside T cannot get the key common to T.

Chapter 3

The Subset-Cover Revocation Framework

3.1 The Framework

Let U be the set of all users. A BE algorithm in this framework defines a collection of subsets $S_1, ..., S_w$, $S_i \subset U$. each subset S_i is assigned a long-lived key L_i. Each user belonging to S_i should be able to deduce the key L_i from the secret information it has.

Given a revoked set $R \subset U$, the remaining users can be partitioned into disjoint subsets $S_{i_1}, ..., S_{i_m}$ so that

$$U \setminus R = \biguplus_{j=1}^{m} S_{i_j}$$

and a session key is encrypted m times with $L_{i_1}, ..., L_{i_m}$.

Specifically, an algorithm in the framework uses two encryption schemes:

- An encryption scheme to deliver the session keys to the privileged users of the disjoint subsets $S_{i_1}, ..., S_{i_m}$ using the long lived keys $L_{i_1}, ..., L_{i_m}$ of those subsets.
- An encryption scheme to encrypt the message itself using the session key.

Components of the Algorithm

Scheme Initiation : Every receiver u is assigned private information I_u such that for all $1 \leq i \leq w$ if $u \in S_i$, I_u allows u to deduce the key L_i corresponding to the set S_i. Also, if $u \notin S_i$ where S_i is any of the sets $S_{i_1}, ..., S_{i_m}$, I_u does not help u in any way to deduce the key L_i.

The Broadcast Algorithm : At the center,

- Choose a session encryption key K
- Given the set R, the center finds out the partition $S_{i_1}, ..., S_{i_m}$ of users in $U \setminus R$.
- It encrypts K with these encryption keys and also encrypts the actual message with K and then broadcasts these.

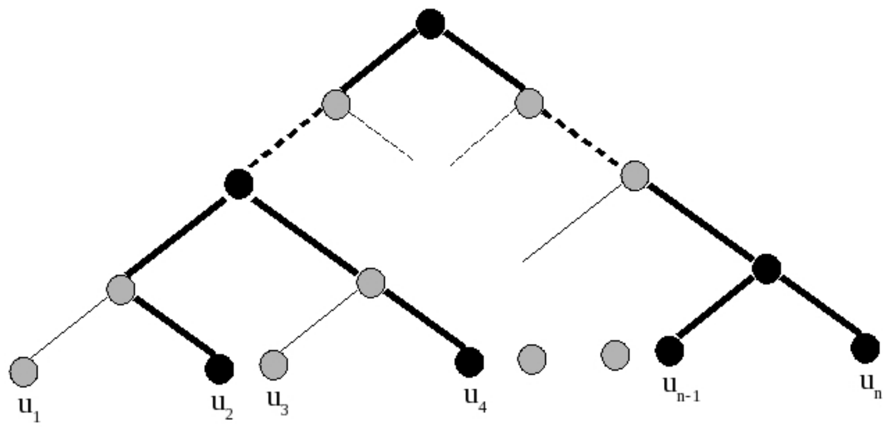


Figure 3.1: The Complete Binary Tree with the revoked nodes and the path from the root to them indicated by dark black lines. When the rest of the nodes and the edges of the tree are deleted, they Form the Steiner Tree as shown in the next figure.

The Decryption Step : At the receiver u,

- Find i_j such that $u \in S_{i_j}$. (If $u \in R$, return NULL)
- Extract the corresponding key L_{i_j} from I_u.
- Decrypt the session key from the corresponding encryption
- Decrypt the actual message using this session key

Security Intuitively, a critical property that can be identified as to be required from the key assignment method in order to provide a secure Subset-Cover algorithm is: ***key-indistinguishability***. An algorithm is said to satisfy this property if for every subset S_i, its key L_i is indistinguishable from a random key given all the information of all the users that are not in S_i.

3.2 Subset-Cover Revocation Schemes

The following are two instantiations of revocation schemes based on the above Subset-Cover framework. Both schemes are $r - flexible$, i.e.; they work with any number of revocations.

3.2.1 The Complete Subtree Method

The collection of subsets $S_1, ..., S_w$ in this scheme corresponds to all complete subtrees in a full binary tree with $|U|$ leaves. Hence, for any node v_i in the full binary tree, let the subset S_i be the collection of receivers u that correspond to the leaves of the subtree rooted at node v_i.

Key Distribution : Assign an independent and random key L_i to every node v_i in the complete subtree. Assign every user u with the $\log|U| + 1$ keys associated with the nodes along the path from root to leaf u.

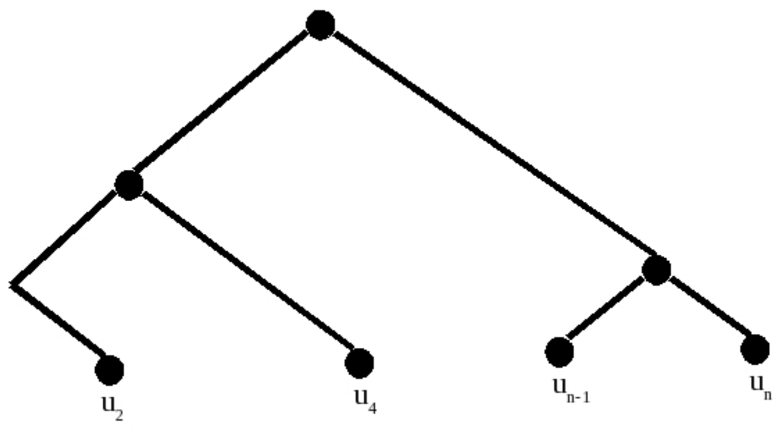


Figure 3.2: The Steiner Tree $ST(R)$ formed from the Complete Binary Tree shown in figure 3.1 by removing all non-revoked nodes and corresponding edges.

Revocation : For a given set R of revoked users, let $u_1, ..., u_r$ be the leaves corresponding to the elements in R. The partitioning of $U \setminus R$ into disjoint subsets would be as follows: Consider the Steiner Tree $ST(R)$ defined as the minimal subtree of the full binary tree that connects all the leaves in R. $ST(R)$ is unique. Let $S_{i_1}, ..., S_{i_m}$ be all the subsets formed by the users at the leaves of the subtrees of the original tree that "hang-off" $ST(R)$.

The following claims are obvious:

- Every leaf $u \notin R$ is in exactly one subset in the above partition.
- A leaf $u \in R$ does not belong to any subset in the above partition.

Theorem 3.2.1. *The complete subtree revocation method requires*

- *message length of at most* $r \log \frac{N}{r}$ *keys,*
- *to store* $\log N$ *keys at the receiver, and*
- $O(\log \log N)$ *operations plus a single decryption operation to decrypt the message.*

where $r = |R|$ *and* $N = |U|$ *keys,*

3.2.2 The Subset Difference Method

The main disadvantage of the Complete Subtree Method is that $U \setminus R$ may be partitioned into a number of subsets that is too large. To see how, consider the path between any two internal nodes of the complete subtree which are neighbours in $ST(R)$ but are quite a distance away in the complete subtree. All subtrees falling off from this path of the complete subtree are individual subsets in the Complete Subtree Method. We intend to keep additional keys for the union of such subsets to reduce the maximum header length.

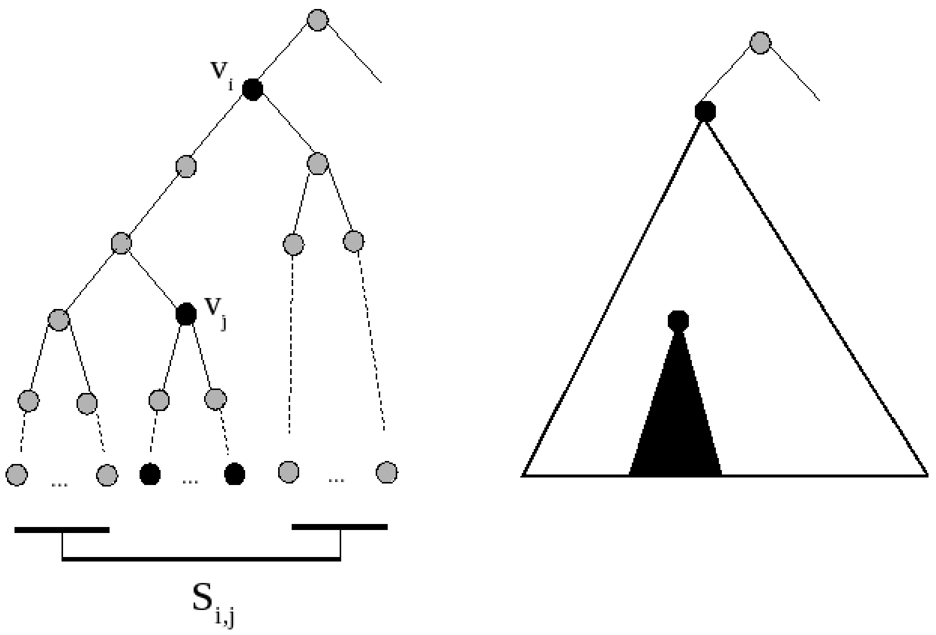


Figure 3.3: Showing the leaves that are in the subset $S_{i,j}$ for the Subset Difference Method

The goal is now to reduce the partition size. We show an improved method that partitions the non-revoked receivers into at most $2|R| - 1$ subsets, thus getting rid of a $\log|U|$ factor and effectively reducing the message length accordingly. In return, the number of keys stored by each receiver increases by a factor of $\frac{1}{2}\log|U|$. In this method any user belongs to substantially more subsets than ($O(|U|)$ instead of $O(\log|U|)$) the Complete Subtree Method and hence the number of keys stored per user increases. The challenge is then to devise an efficient procedure to concisely encode this large set of keys at the user.

We now discuss the key pre-distribution and the covering algorithms. After that we take up an example to illustrate the key distribution amongst users and discuss the resilience.

The Subset Description As in the previous method, the receivers are viewed as leaves in a complete binary tree. The collection of subsets $S_1, ..., S_w$ defined by this algorithm corresponds to subsets of the form $(G1 \setminus G2)$ "a group of receivers G_1 minus another group G_2", where $G_2 \subset G_1$. In the complete binary tree with all users at its leaves, therefore a valid subset is represented by two nodes in the tree (v_i, v_j) such that v_i is an ancestor of v_j. We denote such subset as $S_{i,j}$. A leaf is in $S_{i,j}$ iff it is in the subtree rooted at v_i but not in the subtree rooted at v_j. Figure 3.3 illustrates an example where $S_{i,j}$ is a subset.

The Cover For a given set R of revoked users, let $u_1, ..., u_r$ be the leaves corresponding to the elements in R. The Cover is a collection of *disjoint* subsets $S_{i_1,j_1}, ..., S_{i_m,j_m}$ that partitions $U \setminus R$.

Finding the Cover Let $ST(R)$ be the directed Steiner Tree induced by R and the root. We build the subset collection iteratively, maintaining a tree T which is a subtree of $ST(R)$ with the property that ***"any** $u \in U \setminus R$ **that is below a leaf of** T **has been covered."*** We start by making T to be equal to $ST(R)$. We then iteratively remove nodes from T (while adding subsets to the collection) until T consists of just a single node:

- Find two leaves v_i and v_j in T such that *the least-common-ancestor* v_a *of* v_i *and* v_j *does not contain any other leaf of* T in its subtree. Let v_l and v_r be the two children of v_a such that v_i is a descendant of v_l and v_j is a descendant of v_r. (If there is only one leaf left, make $v_i = v_j$ to be the leaf, v_a to be the root of T and $v_l = v_r = v_a$.)
- If $v_l \neq v_i$ then add the subset $S_{l,i}$ to the collection; likewise, if $v_r \neq v_j$ then add the subset $S_{r,j}$ to the collection.
- Remove from T all descendants of v_a and make it a leaf.

Effectively, the cover algorithm does the following: Find maximal chains of nodes with outdegree 1 in $ST(R)$. For each such chain of length ≥ 1, add a subset to the cover. If the chain is of the form $\{v_{i_1}, ..., v_{i_{l-1}}\}$ (nodes of outdegree 1 in $ST(R)$) followed by a node v_{i_l} which is either a leaf or has outdegree 2, then the subset added to the cover is S_{i_1,i_l}.

To visualize the change from the Complete Subtree Method, we see that there, all subtrees that "hang-off" from the maximal chains between nodes of degree two, used to form distinct subsets. All such nodes are coagulated to form a single subset here in the Subset Difference Method.

Lemma 3.2.2. *Given any set of revoked leaves R, the above method partitions* $U \setminus R$ *into at most* $2|R| - 1$ *disjoint subsets.*

Proof. The Steiner Tree $ST(R)$ has $r = \lceil R \rceil$ nodes as leaves and $r - 1$ nodes as internal nodes. As the algorithm suggests, every node in $ST(R)$ becomes a leaf of the intermediate tree T exactly once. When a node is a leaf, it either gives rise to a new subset or not (when it is an immediate child of v_a in the algorithm). Hence there can be at most $2r - 1$ subsets. □

Key Assignment to Subsets If a receiver needs to store *explicitly*, the keys of all subsets it belongs to, then it would have to store $O(|U|)$ keys. This is because, for each complete subtree T_k it belongs to, user u would have to store a number of keys proportional to the number of nodes in the subtree T_k and *not on the path* from the root of T_k to u. There are $\log N$ such trees one for each height $1 \leq k \leq \log N$, yielding a total of $\sum_{k=1}^{\log N}(2^k - k)$ which is $O(N)$ keys where $N = |U|$.

Instead, we use pseudorandom generators to bring down the total number of keys required to be stored by each user to $O(\log^2 |U|)$. Hence we now define what information I_u the user u must store. While the total number of subsets to which a user belongs to is $O(N)$, these can be grouped into $\log N$ clusters defined by the first subset (from which another subsets is subtracted).

Let $G : \{0,1\}^l \rightarrow \{0,1\}^{3l}$ be a *(cryptographic) pseudorandom sequence generator* similar to the one used in figure 2.1 but whose output length is *three* times the length of the input. Let $G_L(S)$ denote the left third of the output of G on the seed S, $G_M(S)$ the middle third and $G_R(S)$ the right third. Given that any parent node was labeled S, its two children are labeled $G_L(S)$ and $G_R(S)$ respectively where S comes from some original $LABEL_i$ assigned to some ancestor of the children.

Now we choose for each $1 \leq i < N$ corresponding to an internal node in the full binary tree, a random and independent value $LABEL_i$. This value should induce the keys for all legitimate subsets of the form $S_{i,j}$. Consider now the subtree T_i (rooted at v_i). Let $LABEL_{i,j}$ be the label of node v_j derived in the subtree T_i from $LABEL_i$ by repeated application of G using the above method. Following such a labeling, the key $K_{i,j}$ assigned to set $S_{i,j}$ is $G_M(LABEL_{i,j})$.

The process of generating labels and keys for a particular subtree is depicted in Figure 3.4(a). For such a labeling process, given the label of a node v_i, it is possible to compute the labels of all its descendants. Hence it is possible to get all the keys of subsets of the form $S_{i,j}$. On the other hand, without receiving the label of an ancestor of a node, its label is pseudorandom and for a node v_j, given the labels of all its descendants (but not including itself) the key $K_{i,j}$ is pseudorandom ($LABEL_{i,j}$, the label of v_j, is not pseudorandom given this information simply because one can check for consistency of the labels). It is important to note that given $LABEL_i$, computing $K_{i,j}$ requires at most $\log N$ invocations of G.

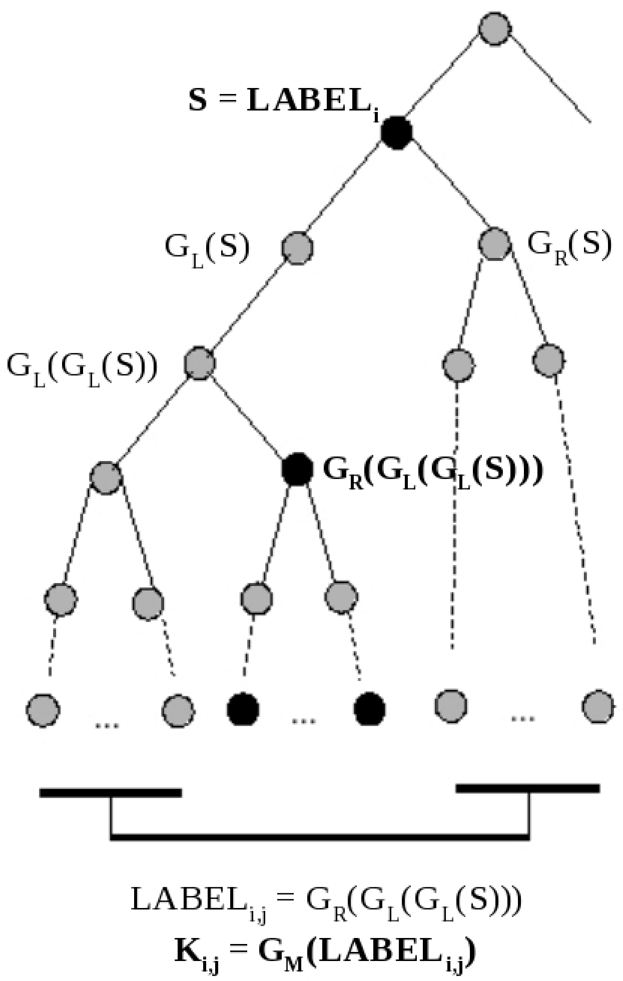
Information I_u held by a user In order to derive the key assignment described above, the information I_u that the user u gets should be such that for each subtree T_i that u is a leaf of, the receiver u should be able to compute $K_{i,j}$ iff j is not an ancestor of u. Consider the path from v_i to u and let $v_{i_1}, \ldots, v_{i_k}$ be the nodes just hanging off the path, i.e. they are adjacent to the path but not ancestors of (see Figure 3.4(b)). Each v_j in T_i that is not an ancestor of u is a descendant of one of these nodes. Therefore if u receives the labels of $v_{i_1}, \ldots, v_{i_k}$ as part of its I_u, then invoking G at most $\log N$ times suffices to compute $K_{i,j}$ for any v_j that is not an ancestor of u.

Number of labels held by a user Total number of labels (from which all keys can be derived) stored by a user u can be found by summing up the labels due to each T_i to which it belongs. Each tree T_i of depth k that contains u, contributes $k-1$ (plus one key for the case where there are no revocations), so the total is

$$1 + \sum_{k=1}^{\log N+1} (k-1) = 1 + \frac{(\log N + 1)\log N}{2} = \frac{1}{2}\log^2 N + \frac{1}{2}\log N + 1.$$

Illustrative Example Let us consider a system with 16 users as shown in figure 3.5 and find out the labels that are assigned to the user u_1. u_1 is part of the subtrees rooted at the internal nodes v_1, v_2, v_4 and v_8. These nodes are the ancestors of u_1 that form a path. The subtrees "falling off" from this path are those rooted at v_3, v_5, v_9, and v_{17} (shown in dark gray). Hence u_1 gets the labels for roots of these subtrees that have been derived from the labels of the ancestors of u_1. The labels derived from $LABEL_1$ are: $LABEL_{1,3}$, $LABEL_{1,5}$, $LABEL_{1,9}$ and $LABEL_{1,17}$. Similarly the labels derived from $LABEL_2$ are: $LABEL_{2,5}$, $LABEL_{2,9}$ and $LABEL_{2,17}$. The labels derived from $LABEL_4$ are: $LABEL_{4,9}$ and $LABEL_{4,17}$. The only label derived from $LABEL_8$ is: and $LABEL_{8,17}$.

Security In order to prove security we show that the key-indistinguishability condition stated in section 3.1 holds for this method, namely that each key is indistinguishable from a random key for all users not in the corresponding subset. Theorem 12 of Section 6 of [NNL01] proves that this condition implies the security of the algorithm.



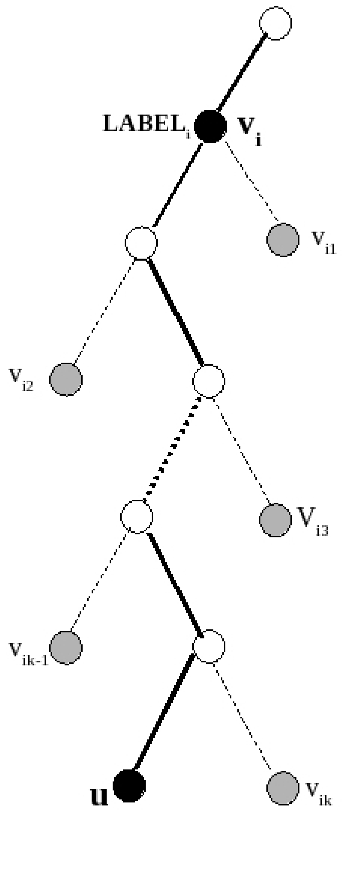


Figure 3.4: (a) The first figure shows derived label of the node v_j of figure 3.3 from the uniform random label $LABEL_i$ of v_i. (b) The second figure shows the labels of $v_{i_1}, \ldots, v_{i_k}$ induced by $LABEL_i$ of v_i that the user u receives.

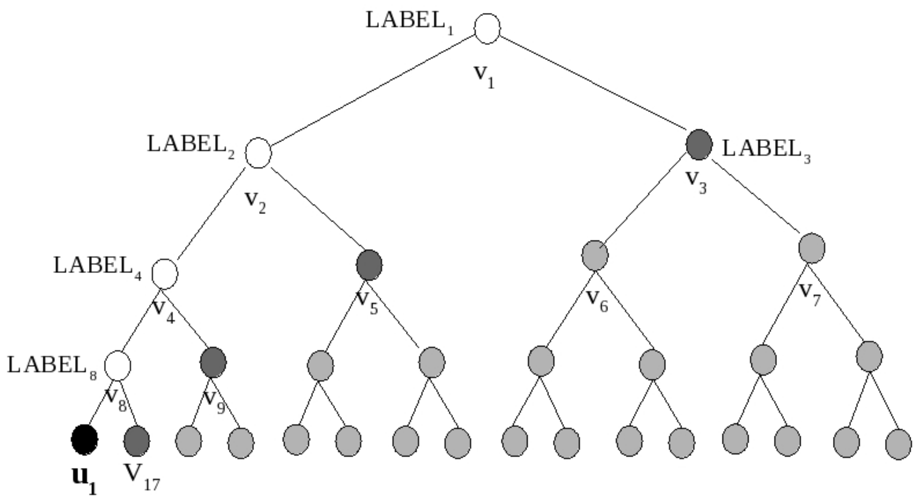


Figure 3.5: An example with 16 users where u_1 gets the following labels: $LABEL_{1,3}$, $LABEL_{1,5}$, $LABEL_{1,9}$, $LABEL_{1,17}$, $LABEL_{2,5}$, $LABEL_{2,9}$, $LABEL_{2,17}$, $LABEL_{4,9}$, $LABEL_{4,17}$, $LABEL_{8,17}$

Observe first that for any $u \in U$, u never receives keys that correspond to subtrees to which it does not belong. Let S_i denote the set of leaves in the subtree T_i rooted at v_i. For any set $S_{i,j}$ the key $K_{i,j}$ is (information theoretically) independent of all I_u for $u \notin S_i$. Therefore we have to consider only the combined secret information of all $u \in S_j$. This is specified by at most $\log N$ labels - those hanging on the path from v_i to v_j plus the two children of v_j - which are sufficient to derive all other labels in the combined secret information. Note that these labels are $\log N$ strings that were generated independently by G, namely it is never the case that one string is derived from another. Hence, a hybrid argument implies that the probability of distinguishing $K_{i,j}$ from any random string of the same length is negligible.

Theorem 3.2.3. *The Subset Difference Method requires*

- *message length of at most* $2r - 1$ *keys,*
- *to store* $\frac{1}{2}\log^2 N + \frac{1}{2}\log N + 1$ *keys at the receiver, and*
- $O(\log N)$ *operations plus a single decryption operation to decrypt the message.*

where $r = |R|$ *and* $N = |U|$ *keys,*

Chapter 4

Layered Subset Difference Algorithms

The paper by Halevy and Shamir [HS02] discussed the Layered Subset Difference Schemes. They first introduced the Basic LSD scheme and then the General LSD scheme.

4.1 The Basic LSD Scheme

Their main observation was the following lemma:

Lemma 4.1.1. *If i, j, k are vertices which occur in this order on some root-to-leaf path in the tree, then $S_{i,j}$ can be described as the disjoint union $S_{i,j} = S_{i,k} \cup S_{k,j}$*

The basic idea of the LSD scheme is to retain only a small subcollection of the $S_{i,j}$ used by the SD scheme. Whenever needed, a discarded set can be replaced by a union of two smaller sets which are in the subcollection.

The subcollections of sets $S_{i,j}$ in the LSD scheme is defined by restricting the levels in which the vertices i and j can occur in the tree. We define some of the $\log n$ levels as *"special"*. The root is considered to be at a special level, and in addition we consider every level of depth $k * \sqrt{\log n}$ for $k = 1, ..., \log n$ as *special.* There are thus $\log n$ special levels which are equally spaced at a distance of $\log n$ from each other. The collection of levels between (and including) adjacent special levels is defined as a *"layer"*.

The subcollection of sets in the LSD scheme is defined in the following way:

Definition 4.1.1. ***Useful Set***: *$S_{i,j}$ is a useful set if it is not empty, and at least one of the following conditions is true:*

- *both i and j belong to the same layer, or*
- *i is at a special level.*

The above definition leads us to the following arguments about dividing the subsets into useful sets as follows: If $S_{i,j}$ is non-empty, j is a strict descendant of i. If they belong to the same layer, then $S_{i,j}$ is a useful set. Otherwise, define k as the first vertex on the path from i to j which is in a special level (possibly i itself). Since i and k are in the same layer, $S_{i,k}$ is a useful set (unless it is empty) even if k and j belong to different layers. Consequently, $S_{i,j}$ is the disjoint union of two useful sets $S_{i,k}$ and $S_{k,j}$ or equal to one of them if the other is empty. This leads us to the following lemma:

Lemma 4.1.2. *Any non-empty set $S_{i,j}$ is either a useful set or the disjoint union of two useful sets.*

So how do we gain? Since the number of covering sets in the SD scheme is bounded by $2r - 1$, and each one of them is replaced by at most two useful sets during the actual broadcast, the number of messages sent by the broadcaster in this scheme is at most $4r - 2$. The $1.25r$ average complexity for r randomly chosen revoked users in the SD scheme suggests a $2.5r$ average complexity in the modified scheme, but in fact there is an additional saving since many of the sets do not get split. Actual experiments on trees with $2,00,000$ users indicate that the average complexity is closer to $2r$, which is only 1.6 times larger than in the original SD scheme. What we gain from this slightly increased message complexity is a significant reduction in the size of the tamper resistant memory in each user's smart card. The analysis follows:

How much do we lose for that? The keys associated with the broadcast sets are generated by the user in the same recursive way as in the SD algorithm. The only labels a user should memorize are those that correspond to sets $S_{i,j}$ in which i is an ancestor of u, j is just hanging off the path from i to u, and the levels of i and j are those specified in the definition of useful sets.

Note that at each level in the tree there can be at most one vertex which can serve as i and one vertex which can serve as j wrt u, and these two vertices are siblings.

To count the number of memorized labels, consider the two possible cases of useful sets:

- *Local sets*: Each layer contains $\sqrt{\log n}$ levels, and thus the number of i and j pairs which belong to that layer is $O((\sqrt{\log n})^2) = O(\log n)$. Since there are $\log n$ possible layers, the number of i and j pairs of this type in the whole tree is $O(\log^{\frac{3}{2}} n)$.
- *Special sets*: Each i in a special level can be associated with a j in any one of the $O(\log n)$ levels underneath it. Since there are $\log n$ possible is, the number of labels of this type is also $O(\log^{\frac{3}{2}} n)$.

Consequently, the total number of labels u has to know is reduced from $O(\log^2 n)$ to $O(\log^{\frac{3}{2}} n)$.

It is easy to show that the choice of $\sqrt{\log n}$ as the distance between consecutive special levels is optimal among all the equidistant partitions. For any choice of distance s between consecutive special levels, the number of keys each user has to remember is $O((s^2 \times \frac{\log(n)}{s}) + (\log n \times \frac{\log n}{s}))$. The first derivative of this expression (as a function of s) is $\log n - \frac{\log^2 n}{s^2}$ which is equal to 0 for

$s = \sqrt{\log n}$. Since the second derivative is positive, choosing this value of s minimizes the storage complexity of this scheme.

The above arguments leads us to the following lemma:

Lemma 4.1.3. *The number of labels memorized by each user u in the basic LSD scheme is* $(\log^{\frac{3}{2}} n)$

4.2 The General LSD Scheme

In this section we show how to further reduce the memory requirements of the user revocation scheme, by solving an interesting graph theoretic problem.

The basic LSD algorithm represents each $S_{i,j}$ as the disjoint union of two sets from a smaller subcollection. It is easy to generalize this observation and represent $S_{i,j}$ as the disjoint union of d sets.

We extend the previous lemma on partitioning the set $S_{i,j}$ of users into a disjoint union of two useful sets by the following idea: We apply a telescoping formula of set differences for any descending chain of sets. The result is the following lemma:

Lemma 4.2.1. *Let $i, k_1, k_2, ..., k_{d-1}, j$ be any sequence of vertices which occur in this order (but not necessarily consecutively) along some root-to-leaf path in the tree. Then*

$$S_{i,j} = S_{i,k_1} \cup S_{k_1,k_2} \cup ... \cup S_{k_{d-1},j}.$$

Mapping the Complete Subtree Method to the Graph Theoretic Problem: Any root to leaf path can be viewed as a line graph of length $\log n$ with directed edges between adjacent vertices. Broadcasting to the set $S_{i,j}$ corresponds to walking from vertex i to vertex j, and addressing all the subtrees that hang off this segment. The original line graph has very few edges (whose labels require very little memory) but these edges provide only a slow way of walking from i to j (with many messages). Since for each original edge the corresponding set is a single subtree, this covering technique is equivalent to the Complete Subtree Method.

Mapping the Subset Difference Technique to the Graph Theoretic Problem: The Subset Difference technique adds to the line graph all the edges in its directed transitive closure. We can now jump from any i to any descendant j in a single jump (and thus address all the users in $S_{i,j}$ with a single message), but each user has to memorize the labels of $O(\log^2(n))$ edges.

Mapping the Basic LSD Scheme to the Graph Theoretic Problem: The basic LSD scheme shows that we only have to use $O(\log^{\frac{3}{2}} n)$ edges (labels) in order to get from any i to any descendant j in two steps (messages).

Motivation for the General LSD Scheme solution: The general LSD scheme considers the following graph theoretic problem: What is the smallest number of edges we have to add to the line graph in order to guarantee the existence of a directed path of length at most d from any i to any descendant j?

To make the graph construction applicable to our user revocation problem, we have to add two additional constraints:

- *Monotonicity*: We can only add an edge from i to one of its descendants.
- *Shrinkage*: If we add an edge from i to j, we also have to add all the edges from i to vertices j between i and j.

We describe an efficient solution to this graph theoretic problem by representing each vertex on the line graph by its distance from the root, expressed as a d-digit number in base $b = O(\log^{\frac{1}{d}} n)$. The root is represented by 0...00, its child is represented by 0...01, etc.

Our goal is to define a small subcollection of *useful transformations* between pairs of numbers which satisfy the monotonicity and shrinkage condition, and allow us to change any i to any larger j with a sequence of at most d useful transformations.

Consider for example the problem of changing $i = 825917$ to $j = 864563$ in standard decimal notation. The simplest solution is to allow arbitrary single-digit transformations such as

$$825917 \rightarrow 865917 \rightarrow 864917 \rightarrow 864517 \rightarrow 864567 \rightarrow 864563.$$

However, these transformations do not satisfy either the monotonicity or the shrinkage condition. Consequently, we have to use a more complicated sequence of transformations defined as follows:

Definition 4.2.1. ***Useful Transformation**: Let i be represented as a d digit number in base b by $\overrightarrow{x}a\overrightarrow{0}$ where $\overrightarrow{x}$ is a sequence of arbitrary digits, $\overrightarrow{a}$ is the rightmost nonzero digit, and $\overrightarrow{0}$ is a sequence of zeroes. The transformation of i to j is called useful if j is represented either by $\overrightarrow{x+1}0\overrightarrow{0}$ or by any number $\overrightarrow{x}a^{'}\overrightarrow{y}$ in which $a^{'} \geq a$ and $\overrightarrow{y}$ is an arbitrary sequence of digits of the same length as 0.*

The basic LSD scheme can be viewed as a special case of this definition for $d = 2$, where two digit numbers ending with 0 are considered to be special. In the general LSD scheme, the number of trailing zeroes in the representation of i determines how special it is and how big is the layer within which it is allowed to jump (j can be any destination between $i + 1$ and the first vertex which is even more special than i, inclusive). In our previous example, these useful transformations allow the broadcaster to split the $S_{i,j}$ set into the following segments:

$$825917 \rightarrow 825920 \rightarrow 826000 \rightarrow 830000 \rightarrow 864563.$$

Note that from the point of view of the broadcaster, the first three transitions are of the first type (which jumps to the end of the layer and increases the specialty level of the vertex), and the last transition is of the second type (which jumps to the middle of the layer).

However, from the point of view of the user he knows the label associated with at most one of these transitions, and its memorized shrunk version is likely to be of the second type even if the broadcast transition was of the first type.

Counting the number of Useful Transformations: Consider any pair of i and j linked by a single useful transformation. We can choose the location of the digit a within i in d ways, and for each location we can choose the $d-1$ digits in the sequences $\overrightarrow{x}$ and $\overrightarrow{y}$ in b^{d1} ways, and the two digits $a \leq a^{'}$ in $\frac{b^2}{2}$ ways. Since $b = O(\log^{\frac{1}{d}} n)$, we get a total number of useful transformations of $O(d * b^{d+1}) = O(d * log^{\frac{(d+1)}{d}} n)$.

Chapter 5

The Punctured Interval Scheme π

The paper by Nam-Su Jho, Jung Hee Cheon, Myung-Hwan Kim and Eun Sun Yoo [JHC$^+$05] discusses the Punctured Interval (PI) Scheme. Their revocation method is based on the Subset-Cover framework proposed by Naor et. al [NNL01].

5.1 Punctured Intervals

Assume that L be a straight line with N dots (users) on it, where N is the number of total users. In our scheme, each user is indexed by an integer $k \in [1, N]$ and he/she is represented by the k-th dot, denoted by u_k, in the line L. Let $p \geq 0$ and $c > 0$ be integers. By a ***p*-punctured *c*-interval** we mean a subset of L which contains c or less consecutive users starting from and ending at non-revoked users and containing p or less revoked users. Let $S_{(p;c)}$ be the set of all p-punctured c-intervals. In each session, the p-punctured c-intervals are to be determined under the following rules:

- The first p-punctured c-interval starts from the leftmost non-revoked user, and each of the following starts from the first non-revoked user after the last non-revoked user of the previous.

- Each p-punctured c-interval contains the maximal possible number of users.

The p-punctured c-interval starting from u_i and ending at u_j with $u_{x_1}, \ldots, u_{x_q}$ revoked users is denoted by $P_{i,j;x_i,\ldots,x_q}$ or $P_{i,j;X}$ in short for $X = \{x_1, \ldots, x_q\}$, where $1 \leq j - i \leq c, 0 \leq q \leq p$, and $i \leq x_1 \leq \ldots \leq x_q \leq j$ if there are revoked users.

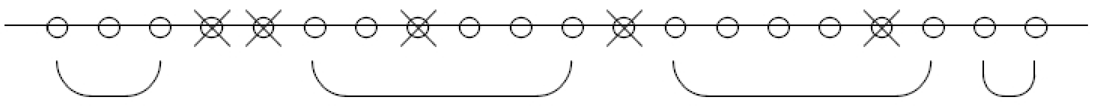


Figure 5.1: 1-punctured 6-intervals

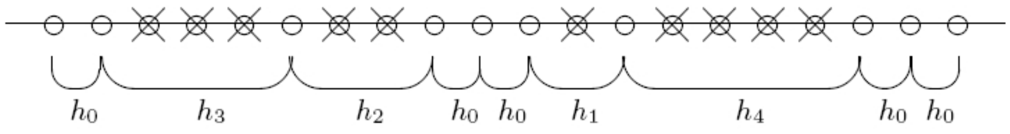


Figure 5.2: Key chain of a 10-punctured 20-interval

5.2 Punctured Interval Scheme $(p; c) - \pi$

We assign just one key to each p-punctured c-interval, which can be easily derived by all non-revoked users in that interval, and construct key chains using one-way permutations in order to reduce the storage size.

Key Generation Let $h_t : \{0,1\}^l \to \{0,1\}^l$ be one-way permutations for $t = 0, 1, \ldots, p$, where l is the key length. To assign one key to each p-punctured interval, we randomly choose N keys $K_{1,1}, K_{2,2}, \ldots, K_{N,N}$ to be given to $u_1, \ldots, u_N$, respectively. From each $K_{i,i}$ the center constructs the one-way key chains under the following rule : For any possible p-punctured c-interval P starting from u_i given,

- The one-way key chain consists only of the keys of all non-revoked users in P. There are no keys of the revoked users in the chain.
- For any non-revoked user $u_k \in P$, if the next user $u_{k+1} \in P$ is also non-revoked, then just apply h_0 to the key of u_k to obtain the key of u_{k+1}.
- If the next t users are revoked and the user $u_{k+t+1} \in P$ is non-revoked, then apply h_t to the key of u_k to obtain the key of u_{k+t+1}, where $1 \le t \le p$.

The following example illustrates how to construct the key chain of a given punctured interval (with $p = 10, c = 20$) : In the key chain of $P = P_{i,j;x_1,\ldots,x_q}$, the key of a non-revoked user $u_k \in P$ is denoted by $K_{i,k;x_1,\ldots,x_t}$, where $i < x_1 < \ldots < x_t < k < x_{t+1} < \ldots < x_q$ and $0 \le t \le q \le p$. For example,

$$K_{5,11} = h_0^6(K_{5,5});$$

$$K_{5,11;7} = h_0^3 h_1 h_0(K_{5,5});$$

$$K_{4,11;5,6,7,9,10} = h^2 h^3(K_{4,4});$$

$$K_{3,11;4,5,7,8} = h_0^2 h_2^2(K_{3,3});$$

$$K_{3,11;4,5,6,7,9} = h^0 h_1 h_4(K3,3); \ldots$$

The center assigns these keys to users so that the user u_k receives $K_{k,k}$ and all possible $K_{i,k;x_1,\ldots,x_t}$s, where $i < x_1 < x_2 < \ldots < x_t < k$ with $0 \le t \le p$ and $2 \le (ki + 1) \le c$. Figure describes the key assignment in the scheme $(3; 5) - \pi$ for u_5:

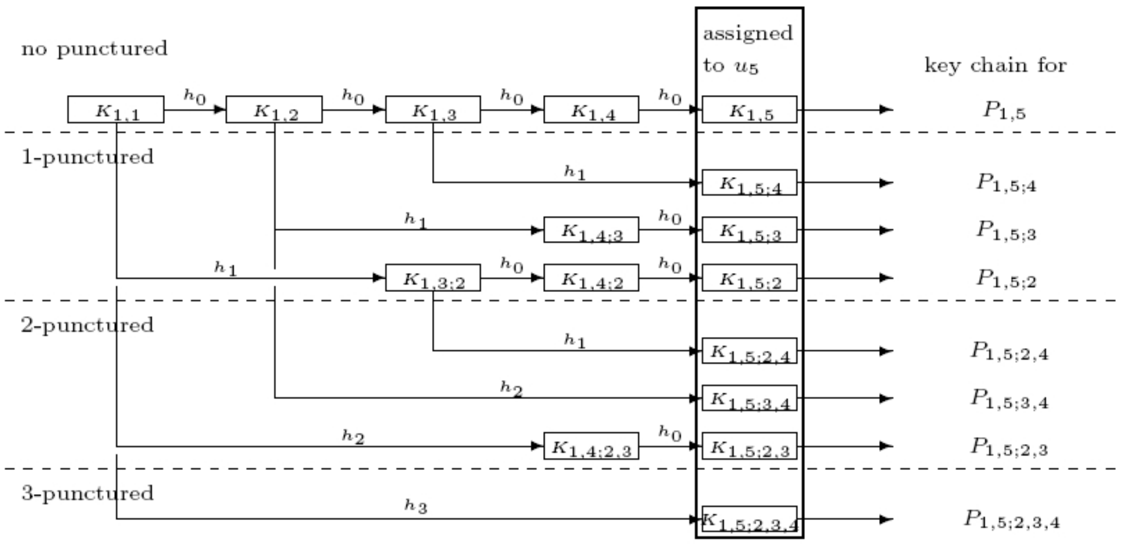


Figure 5.3: One-way key chains starting from $K_{1,1}$, where $c = 5$

Encryption For each session, the center divides L into disjoint p-punctured c-intervals $P_1, \ldots, P_m \in S(p;c)$, whose union covers all the non-revoked users. Let $P = P_{i,j;x_1,\ldots,x_q}$ be one of P_μs. The last key $K_{i,j;x_1,\ldots,x_q}$ of the key chain corresponding to P is called the *interval key* of P. Lets denote the interval key of P_μ by K_{mu} for each $\mu = 1, 2, ldots, m$, just for convenience. Then the center broadcasts :

$$\langle info_1, info_2, \ldots, info_m; E_{K_1}(SK), E_{K_2}(SK), \ldots, E_{K_m}(SK); F_{S_K}(M)\rangle$$

where $info_\mu$ is information on P_μ, the μ-th interval starting from u_{i_μ} and ending at u_{j_μ} with q_μ revoked users. For each μ, $info_\mu$ consists of $i_\mu, l_\mu, l_{\mu,1}, \ldots, l_{\mu,q_\mu}$, where $l_\mu = j_\mu i_\mu + 1$ and $l_{\mu,1}, \ldots, l_{\mu,q_\mu}$ are the distances from u_{i_μ} to the first, ..., to the last revoked users of P_μ, respectively. The starting position i_μ can be represented by $\log N$ bits and the ls are at most $\log c$ bits. So the size of all $info$s is $m(\log N + p \log c)$, which will be ignored when computing the transmission overhead because it is negligible compared to the size of all $E_K(SK)$s.

Decryption Receiving the encrypted message, each non-revoked user u_k first locates the punctured interval that he/she belongs using the $info$s. Let the punctured interval be $P_{i,j;x_1,\ldots,x_q}$, where $i \leq k \leq j$, $k \neq x_1, \ldots, x_q$. Then u_k can find $K_{i,j;x_1,\ldots,x_q}$ as follows:

- Find t for which $x_t < k < x_{t+1}$, where $0 \leq t \leq q$. Here, $t = 0$ and $t = q$ mean that there is no revoked user before and after u_k, respectively.
- Choose $K_{i,k;x_1,\ldots,x_t}$ from the assigned user keys.
- Starting from $K_{i,k;x_1,\ldots,x_t}$, apply one-way permutation h_is under the rule described in Key Generation until the second subscript reaches to j.
- The resulting key is then $K_{i,j;x_1,\ldots,x_q}$.

With this, u_k decrypts $E_{K_{i,j;x_1,\ldots,x_q}}(SK)$ and $E_{SK}(M)$ to obtain the session key SK and the message M, respectively, in order.

Efficiency We analyze efficiency - the transmission overhead (TO), the computation cost (CC) and the storage size (SS) - of the scheme $(p; c) - \pi$. The transmission overhead of the scheme $(p; c) - \pi$ is

$$TO_{(p;c)}(N, r) = \left\lfloor \frac{r}{p+1} \right\rfloor + \left\lceil \frac{N - (p+2)\lfloor \frac{r}{(p+1)} \rfloor}{c} \right\rceil,$$

where N and r are the total number and revoked users, respectively. Especially,

$$TO_{(1;c)}(N, r) = \left\lfloor \frac{r}{2} \right\rfloor + \left\lceil \frac{N - (3)\lfloor \frac{r}{(2)} \rfloor}{c} \right\rceil.$$

This occurs when

```
o x x
```

is repeated from the leftmost user and then the remaining privileged users are on the right, where

```
o
```

and

```
x
```

denote a privileged user and a revoked user, respectively. It is not hard to prove the above equations, but we omit the proof because it is long and tedious. It is trivial that the computation cost $CC_{(p;c)}$ is at most $c1$ computations of one-way permutations, which is independent of N and r. The storage size of each user can be easily computed as follows:

$$SS_{(p;c)} = \sum_{k=0}^{p} \left(\frac{1}{(k+1)} \prod_{i=1}^{k+1} (c-i) \right) + 1,$$

which is also independent of N and r.

Security Note that even a non-revoked user cannot compute the interval keys of the other punctured intervals. Those who do not belong to any punctured interval are the revoked ones and they can never access to the session key. Neither those revoked users who belong to punctured intervals can access to their interval keys because they cannot invert the one-way permutations. The only way to compute the interval key $K_{i,j;x_1,\ldots,x_q}$ of $P_{i,j;x_1,\ldots,x_q}$ is to obtain one of the keys in the key chain. However, no revoked user is assigned a key in the key chain and hence they cannot compute the interval key even though they all collude. Furthermore, the interval keys of previous sessions when the user was not revoked do not help at all in the present session, in which he/she is revoked, because the revocation of him/her results in a totally new key chain.

5.3 Summary

As we are about to finish our study of BE, we try to summarize the most important parameters for the schemes. The schemes of the Subset-Cover Framework and the Punctured Interval scheme give trade-offs between the header size and the number of keys stored at the user. The table 5.1 summarizes these parameters.

Scheme	# Keys at Center	Max. Header Length	# Keys at User
Simple Solutions			
First Simple Solution	$O(n)$	$O(\|P\|)$	$O(1)$
Second Simple Solution	$O(2^n)$	$O(1)$	$O(2^{n-1})$
Zero Message Schemes			
The Basic Zero Message Scheme	$\sum_{i=1}^{k}\binom{n}{i}$	$O(1)$	$\sum_{i=1}^{k}\binom{n-1}{i}$
1-Resilient Scheme based on One-Way Functions	$O(1) \rightarrow (2^{n+1})$	$O(\|P\|)$	$\lceil \log n \rceil \rightarrow (2^{n+1} - \lceil \log n \rceil)$
1-Resilient Scheme based on Computational Number Theoretic Assumptions	$O(n)$	$O(1)$	$n-1$
The Subset Cover Revocation Framework			
The Complete Subtree Method	$2^{\lceil \log n \rceil + 1}$	$\|R\| \log \frac{n}{\|R\|}$	$\log n + 1$
The Subset Difference Method	$2^{\lceil \log n \rceil + 1}$	$2\|R\| - 1$	$O(n) \rightarrow O(\log^2 n)$
The Basic LSD Method	$O(\log^{\frac{3}{2}} n)$	$4\|R\| - 2$	$O(\log^{\frac{3}{2}} n)$
The Punctured Interval Method			
The Punctured Interval Scheme $(p;c) - \pi$	Varries with p and c	$\left\lfloor \frac{r}{p+1} \right\rfloor + \left\lceil \frac{N-(p+2)\left\lfloor \frac{r}{(p+1)} \right\rfloor}{c} \right\rceil$	$\sum_{k=0}^{p}\left(\frac{1}{(k+1)}\prod_{i=1}^{k+1}(c-i)\right) + 1$

Table 5.1: Comparison of the important parameters for the schemes that are part of this study.

Part III

Our Contributions

Chapter 6

Some Basics

6.1 Introduction

The schemes addressing the problem of Broadcast Encryption (BE) work on the common setup of a broadcasting center and a group of users out of which some are *revoked* and the rest are *privileged*. The method for pre-distribution of keys plays an important role in deciding which subsets have keys assigned to them and hence which of them would be counted in to construct the header under a given set of revoked users R. In schemes that try to optimize, some keys are stored at the user and others are generated dynamically from the stored keys and other stored information that is part of I_u for that user u. We begin our thoughts by characterizing our keys in the system. Then we consider a lower bound on the number of keys that should be allotted by the center to the system.

6.2 Key Characterization

We have seen in our study of BE that pseudorandom generators are pretty handy in reducing the information (actual number of keys) that need to be stored at the center or with the users (I_u). In the tree based schemes for example, a random seed is assigned as the label of the root of any subtree of a binary tree and the labels of its two children can all be derived by applying the pseudorandom generator ($G : \{0,1\}^l \rightarrow \{0,1\}^{2l}$) to its label (as shown in figure 2.1). The output of the function G is double the size of its input. The left half of the output is assigned as the label of the left subtree and the right half to the right subtree. The use of a pseudorandom generator ensures that the labels of the subtrees are poly-time indistinguishable from a uniform random string, provided the input to G was also uniformly distributed and random.

Schemes for which the key pre-distribution is devised using pseudorandom generators, use the fact that the subsets of users are correlated in a manner such that labels assigned to those subsets, when concatenated, form the deterministic output of a particular input to the pseudorandom generator. It uses this property to store the labels associated with some superset of users so that the labels of its subsets can be generated dynamically.

Such schemes could be analyzed by assuming that all labels assigned to all subsets of users are uniformly random strings (not generated by pseudorandom generators).

Definition 6.2.1. *Primary Key: A key assigned to a subset of users that is generated from a uniformly random source of strings (and is not an output of any pseudorandom generator).*

Definition 6.2.2. *Derived Key: A key that is a part of the output a pseudorandom generator (the other part is known to be the key for some other subset of users).*

An analysis of a BE scheme can be done based on the assumption that either of the above kind of keys have been assigned to subsets. Practically, this is a time-space trade-off that we need to make. If no pseudorandom generators are used, all labels (keys for subsets) associated with the key-pre-distribution structure have to be stored at the center and user ends. Using them would bring down the space usage, but then the labels have to be generated at run-time as and when required. Usually the later is more convenient since in applications like smart cards memory is scarce, and moreover the fast pseudorandom generators are not difficult to implement.

6.3 Lower bound on the number of Keys

Motivation: One of the important parameters that we try to optimize in every BE scheme is the *number of keys held by each user.* Considering any pair of users, they should never have the same set of keys (or else they can never be mutually revoked). This leads us to the notion that a user would be uniquely identified by a subset of all keys available with the center.

Observation: To be able to revoke all users except one (arbitrary) dynamically, every user should have a unique key unknown to the others. This leads us to the following lemma:

Theorem 6.3.1. *In any BE scheme with total users as n there have to be at least n primary keys in the system to allow any arbitrary revocation.*

The proof of the theorem follows from the argument above. Off course this is not at all a tight lower bound. For most schemes, primary keys are $2n$ or even 2^n for trivial schemes.

Chapter 7

The Hitting Set Framework

All existing schemes of BE consider the set of users U, and try to partition the privileged set $P \subseteq U$ into subsets for which there are pre-assigned common keys so that an encoding of the session key in the header for each such subset in the partition would meet the purpose of broadcasting the message to the entire set of privileged users.

Here, we look at a contrasting perspective to the same problem.

Definition 7.0.1. *Hitting Set Problem: Given a collection C of subsets of a finite set S, positive integer $K \leq |S|$, is there a subset $S^{'} \subseteq S$ with $|S^{'}| \leq K$, such that $S^{'}$ contains at least one element from each subset in C?*

7.1 Does our problem fit the framework?

Now, assume $K = \{k_1, k_2, ..., k_m\}$ to be the set of all *keys* present in the system (generated at the center and distributed amongst the users). Let $U = \{u_1, u_2, ..., u_n\}$ be the set of subsets of K where the subset u_i of keys uniquely identifies the ith *user*. U is the set of all users. Let $R \subseteq U$ be the set of revoked users.

Let $P = \{P_1, ..., P_n\}$ be the set of sets non-revoked keys remaining with the n users. Every user will have at least one unique key that no one else has. So, for privileged user u_i, the set P_i will have at least one element but for a revoked user, u_j, the set P_j will definitely be empty. Hence,

$$P^T = \cup_{P_i \in P} P_i$$

is the collection of all privileged keys.

Given this distribution of keys amongst users (subsets of keys), the system needs to dynamically find the set $K^{'} \subseteq P^T$ of keys such that every nonempty subset in P (privileged user) has at least one element (key) in $K^{'}$. So, the problem is basically of finding the Hitting Set of the set of nonempty subsets in P. Trying to minimize the cardinality of the hitting set is a further extension towards what we would like to achieve.

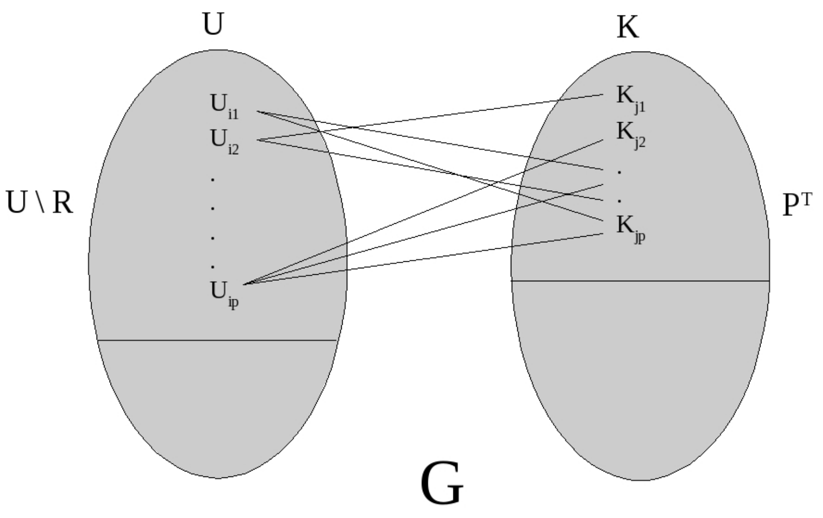


Figure 7.1: The bipartite graph $G((U \setminus R) \cup P^T, E)$ in which, finding the Maximal Matching would give the Hitting Set

We construct a bipartite graph $G((U \setminus R) \cup P^T, E)$ as shown in figure 7.1 whose vertices represent the privileged users in $U \setminus R$ and the privileged keys in P^T and an edge $e_{i,j} \in E$ is an ordered pair (u_i, k_j) where $u_i \in U \setminus R$ and $k_j \in P^T$. Finding the hitting set is same as finding a maximal matching which has polynomial time algorithms.

Chapter 8

The Interval Framework

The key assignment of the existing tree-based schemes have been such that a user used to get unique keys corresponding to a set of subsets having the following containment property: an element of the set of subsets corresponding to a user is either a subset of another element of the set or are mutually exclusive sets of users. This restricted the assignment of keys to non-overlapping subsets of users but gave us the advantage of associating the subsets with nodes of a tree and hence conveniently using a PRG for reducing the number of keys to be stored at a user's end. The associated bound on the header size was $2r - 1$ where R is the set of revoked users and $r = |R|$.

We propose the Interval Framework, we move to an idea where a user would be getting keys corresponding to subsets that would be overlapping but -unlike for tree-based schemes- may not be completely contained in each other. For key pre-distribution, the center assumes ***a circular ordering*** of the users. Let U be the set of users that are assigned the order $u_0, \ldots, u_{n-1}$, where $n = |U|$.

Definition 8.0.1 (Interval)**.** *A subset of **consecutive** users in the assumed cyclic order is called an **interval**.*

8.1 Basic Circular Order Scheme

Keeping the first simple solution stated in section 1.2 in mind, where every user is assigned a unique key, we look at a step ahead to reduce the header size by doubling the number of keys. Every user is still assigned a unique key. Along with that, every adjacent pair is assigned a unique key. So, a user effectively gets three keys: one that is unique to itself and two other which uniquely identify its pairing with its two neighbours in the cyclic order. The total number of keys with the center is $2n$. The header size is at most $n - r$ and is at least $(n - r)/2$.

8.2 All-Interval Circular Order Scheme

In the basic scheme, we considered intervals of length one and two only starting at a user and had assigned keys to those intervals only. In the all-interval circular order scheme, we consider all intervals starting at a user u_i in this circular array of users, with the lengths of the interval l varying from 1 to $(n - 1)$. Hence, there will be $(n - 1)$ such intervals for every user. Hence, for n users, the total number of such intervals would be $n(n - 1)$.

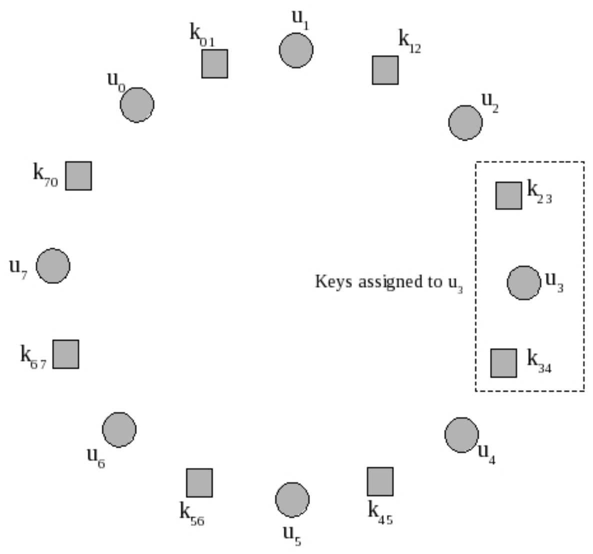


Figure 8.1: An example set of keys for the Basic Circular Order Scheme. Nodes u_0 to u_7 correspond to unique keys assigned to the users. Nodes k_{01} to k_{70} represent keys assigned to pairs of users. Key $k_{i,i+1}$ is the key assigned to the user u_i and its neighbour u_{i+1}.

The number of intervals a user is part of (and hence, the number of keys assigned to every user) is: $n(n-1) - (n-1)(n-2)/2$, which is $O(n^2)$.

Where we gain is the header size. The best known upper bound on the header size is $2|R| - 1$. In this case it would be $|R|$.

Security Notion: Since the two schemes above are such that the keys assigned to subsets are independent uniform random strings (primary keys), hence the security notion is information theoretic. Both the schemes are fully resilient against traitors i.e.; even if an adversary gets hold of the keys corresponding to a subset of users (the traitors), it will not be able to derive any information about the keys corresponding an interval not containing any one of the traitors.

8.3 The Layered DAG of Intervals Scheme

Although the all-interval circular order scheme achieves a better header size, the number of keys that a user has to carry is $O(n^2)$. We intend to bring this down by trying to draw a relation between overlapping intervals and represent it by a *directed acyclic graph* and hence assign keys using it.

8.3.1 Construction of the Layered DAG of Intervals

Identifying the Nodes of the DAG: The all-interval circular order scheme identifies all the possible $n(n-1)$ intervals in the cyclic order. We use each intervals to label a node of our *directed acyclic graph*. Hence, our graph has $n(n-1)$ nodes. We would call the nodes as intervals interchangeably, whenever appropriate.

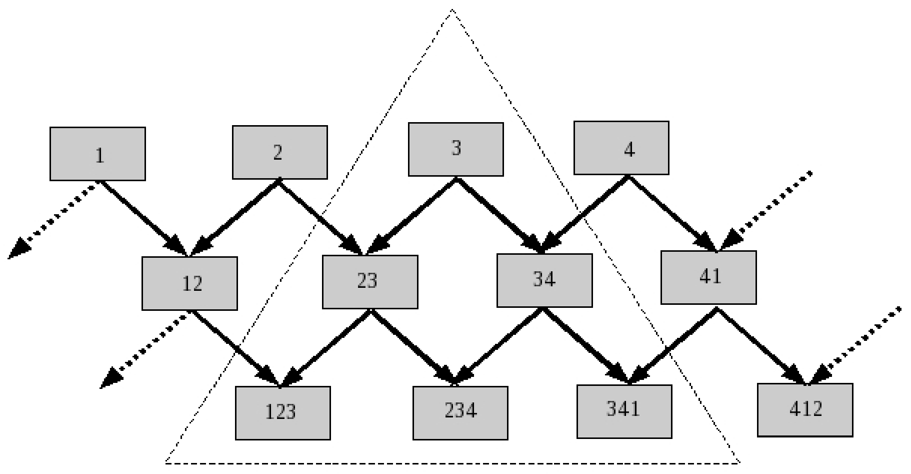


Figure 8.2: An example L-DAG Scheme for four users. There are three layers of intervals of length 1, 2 and 3 each. Each interval corresponds to a node in the directed graph.

Layering: We first arrange the intervals according to their lengths into layers. The first layer has all the n intervals of length 1. The second layer has all the n intervals of length 2. We proceed in a similar manner to construct all the $(n-1)$ layers. The last layer would contain the n intervals of length $(n-1)$ each.

Edge Assignment: If we look at the arrangement of the nodes into layers, then we see that the union of two consecutive intervals at layer i (the layer with intervals of length i) is an interval in layer $(i+1)$. We assign two directed edges from the two intervals of layer i to the intervals in layer $(i+1)$.

8.3.2 Key pre-distribution

Key pre-distribution at the center: All the nodes at layer 1 (each corresponding to a singleton set of a user), is assigned a primary key. A Pseudo-Random Generator of the form $G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ that doubles a uniform random string is used to label the edges coming out of the nodes. Nodes at layer 2 are assigned a key formed by a simple XOR of the labels of its incoming edges. Again, the edges coming out of the nodes of the layer 2 are labeled by applying the keys of the nodes as inputs to the PRG and so on.

Key pre-distribution to a user: A user has to receive the keys of all the intervals to which it belongs (not only keys of intervals that start at that user). Let us consider the intervals to which a particular user u_i belongs. It is a subgraph of the DAG described above with, with a triangular planar embedding. From the construction of the DAG and the key-assignment method explained above, we see that each user needs to store $2n-3$ keys corresponding to the two sides (of length $(n-1)$ each) of the triangle to derive the keys of the rest of the intervals (inside the subgraph) to which it belongs.

8.3.3 Resilience:

Assuming computational security, a user only has the keys of intervals in the triangular planar subgraph corresponding to it. Even if an adversary gets hold of the keys of a number of such users (traitors), it can not by any means derive any other key that it has already not caught hold of (from the traitors). Hence, this scheme is also fully resilient.

8.4 The Inverted Layered DAG of Intervals Scheme

Using the idea of the the layered DAG of intervals scheme, we modify the construction a bit to achieve far better results.

8.4.1 Construction of the Inverted Layered DAG of Intervals

The DAG construction in this case is same in terms of the association of intervals with nodes and layering. The edge assignment is also similar but with inverted directions of the edges. Precisely, if we reverse the direction of all the edges of the previous DAG, then we get the graph for this scheme.

8.4.2 Key pre-distribution

We form a tree T such that the nodes at layer $(n-1)$ of the DAG above are the leaves of the tree. The root of this tree is assigned a primary key. This is the only primary key the center has to store.

Key pre-distribution at the center: The keys now assigned to the intervals of layer $(n-1)$ are hence derived from the above tree. Now, for rest of the nodes of the DAG, the outgoing edges of each node are assigned a label using the PRG G as before. The intervals at layer $(n-2)$ is assigned a key by XOR-ing the labels of their incoming edges and so on.

Key pre-distribution to a user: A user is a part of $(n-1)$ intervals out of the n intervals of layer $(n-1)$ of the DAG. So, it has to have the keys of all these $(n-1)$ intervals. Now, if we consider these $(n-1)$ intervals as leaves of the tree T, then keys of $\log n$ nodes of T are sufficient to generate all these keys.

8.4.3 Resilience:

This scheme is 1-resilient. An adversary having the keys of any two users, can derive the keys of any other user since they will have all the n keys associated with the intervals at layer $(n-1)$.

8.5 Summary of parameters of Interval Framework:

Table 8.1 summarizes the parameters of the different schemes of the Interval Framework.

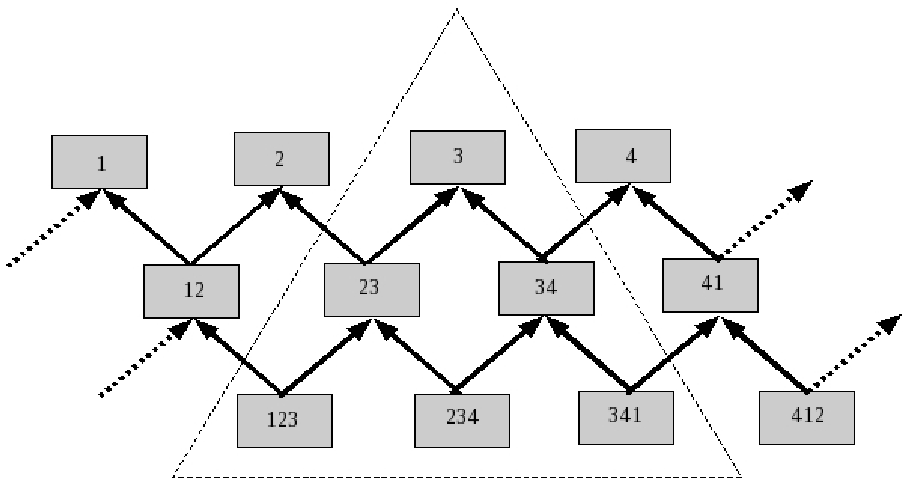


Figure 8.3: An example Inverted L-DAG Scheme for four users. There are three layers of intervals of length 1, 2 and 3 each. Each interval corresponds to a node in the directed graph. The edges here are exactly opposite to the ones in L-DAG.

Scheme	# Keys at Center	Header Length	# Keys at User	Resilience
First Simple Solution	$O(n)$	$O(\|P\|)$	$O(1)$	$FullyResilient$
The Basic Zero Message Scheme	$2n$	$n-r$	3	$FullyResilient$
All-Interval Circular Order Scheme	$n(n-1)$	r	$n(n-1)-(n-1)(n-2)/2$	$FullyResilient$
Layered DAG of Intervals Scheme	n	r	$2n-3$	$FullyResilient$
Inverted Layered DAG of Intervals Scheme with tree support	1	r	$\log n$	$1-resilient$

Table 8.1: Summary of the parameters of schemes of the Interval Framework

8.6 Comparison with the Punctured Interval Scheme $(p; c) - \pi$ and Improvements

It turns out that our Interval Framework results (through the L-DAG Scheme) are just the same as the PI framework discussed in chapter 5. Since we consider intervals which are not punctured, $p = 0$ and $c = N - 1$ would result in the header size $r + 1$ while our result gives header size r. On the other hand, in our scheme a user needs to store $2N - 3$ keys while in the $(0; N - 1) - \pi$ scheme, the storage size is N.

We can modify our scheme a bit by keeping just $N/2$ layers in the L-DAG instead of $N - 1$. This would result in a header size of at most $r + 1$ and number of keys required to be stored by each user would be $N - 1$. This is almost equivalent to the $(0; N - 1) - \pi$ scheme. But our scheme has an advantage over the $(0; N - 1) - \pi$ scheme in the fact that we use pseudorandom generators in our scheme which are easily implemented, while the $(0; N - 1) - \pi$ scheme uses one-way permutations. Now, all known methods for implementing one-way permutations are time-consuming. A one-way permutation can be constructed using number theoretic operations, such as, x going to $g^x \bmod p$ for some prime p. Such operations, however, are significantly slower than symmetric key primitives. There is no natural way of constructing a one-way permutation from symmetric key primitives. For example, a block cipher provides a permutation for each value of the key K. But, then the permutation is not publicly known; if K is known, then it is easy to invert a block cipher. Similarly, there are no natural ways of obtaining a one-way permutation from stream ciphers or hash functions. In view of the above, to implement the PI construction, one will require expensive number-theoretic operations. Further, for the one-wayness to be guaranteed, the size of the underlying group should be quite large, at least 1024 bits. The number of keys per user (with $c = N - 1$ and $p = 0$) is $(N - 1)$, but, these are 1024-bit keys. So that the total key size is $1024(N - 1)$ bits. This is where our proposed scheme has an advantage over the $(0; N - 1) - \pi$ scheme. But again, instead of using one-way permutations, the $(0; N - 1) - \pi$ scheme can use a length-doubling pseudorandom generator as shown in figure 2.1 and just consider the first l bits of it as its output. This will as well work for the $(0; N - 1) - \pi$ scheme.

But what we get from this is an idea for extending our L-DAG scheme to get a tradeoff between reducing the keys stored at users by a factor of k while increasing the header size by k. In general, the L-DAG scheme can be modified to keep only N/k layers in the L-DAG used in the scheme. This would result in a header size of at most $r + k - 1$ and number of keys required to be stored by each user would be $\frac{2N}{k} - 1$.

Chapter 9

The k-ary Tree Schemes

9.1 Motivation

In the Complete Subtree method a user gets keys assigned to all nodes on the path from the root of the binary tree to the leaf corresponding to itself. In the Subset Difference method, a user gets labels assigned to nodes "falling off" from the path from the root to the leaf corresponding to the user. The Subset Difference Method assigns keys to subsets that were assigned keys in the Complete Subtree Method and more. The combination of these two concepts of key assignment come to use when the tree that we construct out of the users is ternary or k-ary in general.

9.2 The Ternary Tree Scheme

Ternary trees intuitively, should be able to reduce the number of keys that need to be stored at the user by reducing the height of the tree for a fixed number of users (as compared to binary trees) and also allowing the use of a pseudorandom generator that will expand a random label of a node four times its length. The key assignment to subsets for such a tree has to be done in such a manner that revocations do not give rise to too many subsets.

9.2.1 Key Assignment

The users are arranged as leaves of a complete ternary tree (each node has three or no children). We start with this assumption for a number of users of the form 3^m. Later on we can extend the same idea for any number of users in the system. We assume the nodes of the ternary tree that are at the same distance from the root to form layers of nodes. The root node is assumed to be in layer 0.

Primary Labels

Each node of the ternary tree is assigned *two* independent random labels called **primary labels** : node v_i is assigned $LABEL_i^{cs}$ and $LABEL_i^{sd}$. The label $LABEL_i^{cs}$ is assigned to be used as a Complete Subtree label while $LABEL_i^{sd}$ is assigned to be used as a Subset Difference label.

Pseudorandom Generators for Derived Labels

Other than the uniformly random labels assigned to nodes, the nodes also get derived labels from the Subset Difference labels of its ancestors. The derived label of a child of v_i is generated using the pseudorandom generator $G : \{0,1\}^l \rightarrow \{0,1\}^{4l}$. Let s (string of length l) be the Subset Difference label (can be derived or primary) of v_i. An input s to the pseudorandom generator G is expanded four times its length. This output is split into four parts of length l each namely, $G_k(s)$, $G_l(s)$, $G_m(s)$ and $G_r(s)$ i.e.;

$$G(s) = G_k(s)||G_l(s)||G_m(s)||G_r(s).$$

$G_l(s)$, $G_m(s)$ and $G_r(s)$ are the derived labels of the left, middle and the right children of v_i which was assigned the the label s. $G_k(s)$ is used as a key for the subset $S_{p,i}$ by the subset difference method where v_p is the parent of v_i whose primary label $LABEL_p^{sd}$ has been used to get the derived label s of v_i.

Now, if we consider a user u_i at the leaf level of this ternary tree, it is given labels of the following nodes:

- Labels corresponding to the Subset Difference method: the derived labels of all nodes “falling off” from the path from the root to itself generated from the primary labels of all its ancestors in the path.

- Labels corresponding to the Complete Subtree method: the primary labels of all its ancestors in the path from the root to itself (the leaf).

Counting total number of subsets with keys

Key by the Complete Subtree method: Every node gets a primary label that acts as the key for the subset composed of all users that are leaves to that subtree.

Key by the Subset Difference method: It assigns a node at the ith layer the labels derived from (at most $\lceil \log_3(n) \rceil$) primary labels of its ancestors. When each such label s is applied as an input to the pseudorandom generator G, the output has $G_k(s)$ which acts as a key for some subset. So, each such node at the ith layer gets (i) such labels.

Total Keys: A node at the ith layer has $(i) + 1$ associated keys. There are 3^i nodes at layer i. Hence the total number of keys used in the system is:

$$\sum_{i=1}^{\lceil \log_3(n) \rceil} (i+1)(3^i) = \sum_{i=1}^{\lceil \log_3(n) \rceil} (i3^i) + \sum_{i=1}^{\lceil \log_3(n) \rceil} (3^i).$$

Note 9.2.1. *The subsets getting keys by the Complete Subtree method are all complete subtrees and the subsets getting keys by the Subset Difference method are strictly not complete other than the singleton subsets at the leaf level. Hence we claim that the subsets to which keys are assigned and have cardinality more than one, get keys by exactly one of the methods and not by both. For the singleton subsets at the leaf level, we will consider the keys assigned by the Complete Subtree method.*

Keys per User

A user in this method gets $\lceil \log_3(n) \rceil$ keys by the Complete Subtree method and $\sum_{i=1}^{\lceil \log_3(n) \rceil} i$ by the Subset Difference method which totals to:

$$\lceil \log_3(n) \rceil + \frac{(\lceil \log_3(n) \rceil)(\lceil \log_3(n) \rceil + 1)}{2}.$$

9.2.2 Subset Cover

We now construct an algorithm that takes as input the set of revoked users, and finds the minimal subset cover using the subsets to which keys have been assigned by the key assignment method above. The cover that we get for an arbitrary revocation results in the minimum possible header for that revocation using the available subsets with assigned keys.

Cover Algorithm

The algorithm goes as follows:

- Construct the Steiner Tree $ST(R)$ induced by the set of revoked users R from the complete ternary tree using the nodes only on the paths starting from the root to the leaves corresponding to the revoked users.
- We build the subset cover by constructing an intermediate tree T that is initially equal to the Steiner Tree $ST(R)$.
- We keep iteratively removing nodes from this tree T and their corresponding subtrees (assuming that all leaves under this subtree have been covered) and keep adding corresponding subsets to the cover by the following algorithm:
  - Find a node v_i in T such that it is the *least-common-ancestor* of leaves in T and has no internal node in its subtree. Let v_i have l (can take values $1, 2$ or 3) children namely $v_{j_1}, \ldots, v_{j_l}$. Let $v_{{j_1}'}, \ldots, v_{{j_l}'}$ be the children of v_i in the original ternary tree which are ancestors of the l nodes of v_i.
  - For all these leaf nodes v_{j_k} of v_i which are not the same as their ancestor nodes $v_{{j_k}'}$, add subsets $S_{{j_k}', j_k}$ to the cover. Depending on the value of l, there will be three cases as follows:
    - $l = 1$ which can happen if there is only one leaf left in T rooted at v_{root}, in which case, S_{root,j_1} is added to the cover. This subset has a key assigned by the Subset Difference Method.
    - $l = 2$ which means there are two children v_{j_1} and v_{j_2} of v_i in T. In this case, S_{i,j_1} and S_{i,j_2} are added to the cover. These subsets have keys assigned by the Subset Difference Method. The set corresponding to the leaves of the third child subtree is also added to the cover and it has a key assigned through the Complete Subtree Method.
    - $l = 3$ which means there are three children v_{j_1}, v_{j_2} and v_{j_3} of v_i in T. In this case, S_{i,j_1}, S_{i,j_2} and $S_{i,j_3}$are added to the cover. These subsets have keys assigned by the Subset Difference Method. No other set needs to be added to the cover.

Maximum Header Size

There are two types of nodes in $ST(R)$: *leaf nodes* (corresponding to revoked users), and *internal revoked nodes* of degree two or more. The root node may be one of the internal revoked nodes. If not, then it is a special third kind of node of degree one. On analyzing the above algorithm carefully, we make the following claims:

Lemma 9.2.1. *A leaf node in T ($ST(R)$ to begin with) gives rise to at most one subset due to the above cover algorithm.*

Proof. If T has only one leaf (that would imply that there are only two nodes left: the leaf and the root), that would give rise to a single subset to cover all privileged users (that have not yet been covered) by the Subset Difference method. Now, if T has more than one leaf and one of them is (say) v_{j_1}, then its corresponding $v_{{j_1}'}$ in the above algorithm gives rise to one set of the cover $S_{{j_1}',j_1}$ if $v_{{j_1}'} \neq v_{j_1}$. If $v_{{j_1}'} = v_{j_1}$ then it does not give rise to any subset of the cover. □

Lemma 9.2.2. *An internal revoked node other than the root in $ST(R)$ has degree at least two. The root can have degree one.*

Proof. An internal revoked node (other than the root) in $ST(R)$ has been formed because there are at least two revoked nodes (internal or leaf) in its subtree. □

Lemma 9.2.3. *An internal revoked node of degree two in $ST(R)$ gives rise to at most two subsets due to the above cover algorithm.*

Proof. An internal revoked node of degree two in $ST(R)$ has three subtrees in the original complete subtree: two with some revoked leaves in each and one without any revoked leaf. At some point of time in the algorithm, when it becomes the node v_i, it will have two leaves. The subtree of this v_i without any revoked leaf in the original ternary tree is one that occurs due to this internal revoked node. The other two subtrees have one revoked leaf each in T and hence subsets occurring due to them have been taken care of by the algorithm and the subtrees get removed before this internal revoked node. These are counted as subsets generated by the two nodes which are leaves of v_i. Now, when this node v_i becomes a leaf in T at some stage, it may give rise to another subset which has been assigned a key through the Subset Difference Method. □

Lemma 9.2.4. *An internal revoked node of degree three in $ST(R)$ gives rise to at most one subset due to the above cover algorithm.*

Proof. When such a node of degree three becomes the v_i of the above algorithm with all its three leaves revoked, all its subtrees gets deleted and it itself becomes a leaf. Upon becoming a leaf itself, it can give rise to at most one subset to be added to the cover that would have been assigned a key through the Subset Difference Method. □

Lemma 9.2.5. *An internal revoked node in $ST(R)$ gives rise to at most two subsets due to the above cover algorithm.*

Proof. Follows from lemma 9.2.3 and 9.2.4. □

Theorem 9.2.6. *The above cover algorithm gives rise to a header size of at most* $3|R| - 2$.

Proof. Follows from lemma 9.2.1 and 9.2.5. □

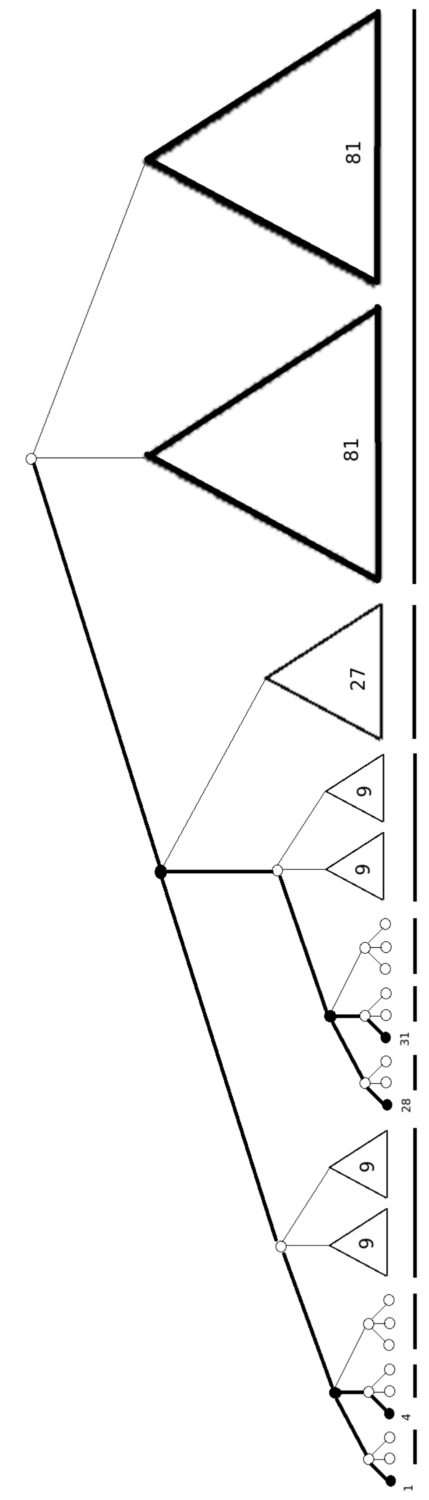


Figure 9.1: Example revocation in Ternary Tree scheme

9.2.3 Revocation and Resilience

Resilience: It can be easily observed that any user gets keys only for those subsets to which it belongs. The proof for the fact that this scheme is fully resilient follows the same lines as the security argument stated in section 3.1 based on the key-indistinguishability condition. It can be shown that each key is indistinguishable from a uniform random string for all users not in the corresponding subset.

Revocation: Any user can be revoked in the proposed scheme.

9.2.4 Summary

This is a scheme based on the Complete Subtree and the Subset Difference schemes needing $\frac{3}{2}$ times the header size of the Subset Difference scheme but reducing the keys per user by a factor of $(\log_3 2)^2$. While the Subset Difference method uses

$$\frac{(\lceil \log_2(n) \rceil)(\lceil \log_2(n) \rceil + 1)}{2}$$

keys, the Ternary Tree scheme uses

$$\lceil \log_3(n) \rceil + \frac{(\lceil \log_3(n) \rceil)(\lceil \log_3(n) \rceil + 1)}{2}$$

keys for header size around $\frac{3}{2}$ times of the former.

9.3 The k-ary Tree Scheme

9.3.1 Motivation

The Ternary Tree scheme of section 9.2 draws us to an intuition which can lead us to an improvement on the number of keys of the Ternary Tree Scheme keeping the header size $3r - 2$. The internal nodes of the Steiner Tree $ST(R)$ have subtrees that may or may not have revoked leaves. If a particular subtree has revoked leaves, each such subtree can be made to result into subsets by the subset difference method as we did for the ternary tree. But what about the subtrees that do not have any revoked leaf in them? If we allow such subtrees to form more than one subset, then we cannot restrict the number of subsets due to internal nodes to two and hence the header size will definitely not remain $3r - 2$. Hence all such combinations of possibly unrevoked set of subtrees have to be allotted independent keys.

k-ary trees again, should be able to reduce the number of keys that need to be stored at the user by further reducing the height of the tree than the Ternary Tree for a fixed number of users and also allowing the use of a pseudorandom generator that will expand a random label of a node $k + 1$ times its length. As before, the key assignment to subsets for such a tree has to be done in such a manner that revocations do not give rise to too many subsets.

9.3.2 Key Assignment

The users are arranged as leaves of a complete k-ary tree (each node has k or no children). We start with this assumption for a number of users of the form k^m. Later on we can extend the same idea for any number of users in the system. We assume the nodes of the k-ary tree that are at the same distance from the root to form layers of nodes. The root node is assumed to be in layer 0.

Primary Labels

Each node of the k-ary tree is assigned *two types of* independent random labels called **primary labels** : node v_i is assigned $LABEL_i^{sd}$ which will be used for the Subset Difference label. It also gets for each layer above it, labels $LABEL_i^{cs_1}, \ldots, LABEL_i^{cs_d}$ assigned to be used as a Complete Subtree label where $d = \sum_{i=0}^{k-3} \binom{k-1}{i} = 2^{k-1} - (k-1) - 1$. Each such label will be used as a key for a subset formed by union of leaves of unrevoked subtrees. There are a total of $\sum_{i=0}^{k-2} \binom{k}{i} = 2^k - k - 1$ such keys associated to subtrees in each layer above it on its path or "hanging off" it.

Pseudorandom Generators for Derived Labels

Other than the uniformly random primary labels given to nodes, the nodes also get derived labels from the Subset Difference labels of its ancestors. The derived label of a child of v_i is generated using the pseudorandom generator $G : \{0,1\}^l \rightarrow \{0,1\}^{(k+1)l}$. Let s (string of length l) be the Subset Difference label (can be derived or primary) of v_i. An input s to the pseudorandom generator G is expanded $(k+1)$ times its length. This output is split into $(k+1)$ parts of length l each namely, $G_k(s), G_{l_1}(s), \ldots, G_{l_{k-1}}(s)$ and $G_{l_k}(s)$ i.e.;

$$G(s) = G_k(s)||G_{l_1}(s)|| \ldots ||G_{l_{k-1}}(s)||G_{l_k}(s).$$

$G_{l_1}(s), \ldots, G_{l_k}(s)$ are the derived labels of the k children of v_i which was assigned the the label s. $G_k(s)$ is used as a key for the subset $S_{p,i}$ by the subset difference method where v_p is the parent of v_i whose primary label $LABEL_p^{sd}$ has been used to get the derived label s of v_i.

Now, if we consider a user u_i at the leaf level of this k-ary tree, it is given labels of the following types:

- Labels corresponding to the Subset Difference method: the derived labels of all nodes "falling off" from the path from the root to itself generated from the primary labels of all its ancestors in the path.
- Labels corresponding to the Complete Subtree method: the primary labels of all its ancestors in the path from the root to itself (the leaf) and also corresponding to the union of subtrees of siblings of its ancestors.

Counting total number of subsets with keys

Keys by the Complete Subtree method: Every internal node gives rise to $\sum_{i=0}^{k-2} \binom{k}{i} = 2^k - k - 1$ primary keys as explained above that act as keys for union of subsets composed of all users that are leaves to the unrevoked subtrees of that node. There are k^i nodes in the ith layer.

So the total number of keys due to the internal nodes by the Complete Subtree Method comes down to:

$$\sum_{i=0}^{\lceil \log_k N \rceil - 1} k^i(2^k - k - 1).$$

Since the leaf nodes do not have any subtrees, they do not need any key to be assigned by the Complete Subtree Method.

Keys by the Subset Difference method: It assigns a node at the ith layer the labels derived from (at most $\lceil \log_k(n) \rceil$) primary labels of its ancestors. When each such label s is applied as an input to the pseudorandom generator G, the output has $G_k(s)$ which acts as a key for some subset. So, each such node at the ith layer gets (i) such labels.

Total Keys: A node at the ith layer $(0 \le i < \lceil \log_k(n) \rceil)$ has $(i + \sum_{i=0}^{k-2} \binom{k}{i}) = (2^k - k - 1 + i)$ associated keys and for $i = \lceil \log_k(n) \rceil$, there are (i) keys. There are k^i nodes at layer i. Hence the total number of keys used in the system is:

$$\sum_{i=1}^{\lceil \log_k(n) \rceil} ik^i + \sum_{i=0}^{\lceil \log_k(n) \rceil - 1} k^i(2^k - k - 1).$$

Note 9.3.1. *Here again, the subsets getting keys by the Complete Subtree method are all complete subtrees and the subsets getting keys by the Subset Difference method are strictly not complete other than the singleton subsets at the leaf level. Hence we claim that the subsets to which keys are assigned and have cardinality more than one, get keys by exactly one of the methods and not by both. For the singleton subsets at the leaf level, we will consider the keys assigned by the Complete Subtree method.*

Keys per User

A user in this method gets $\sum_{i=0}^{k-3} \binom{k-1}{i} = (2^{k-1} - (k-1) - 1)$ keys per layer and hence $(2^{k-1} - k)(\lceil \log_k(n) \rceil)$ keys by the Complete Subtree method and $\sum_{i=1}^{\lceil \log_k(n) \rceil} i$ by the Subset Difference method which totals to:

$$(2^{k-1} - k)(\lceil \log_k(n) \rceil) + \frac{(\lceil \log_k(n) \rceil)(\lceil \log_k(n) \rceil + 1)}{2}.$$

9.3.3 Subset Cover

We now construct the algorithm that takes as input the set of revoked users, and finds the minimal subset cover using the subsets to which keys have been assigned by the key assignment method above. The cover that we get for an arbitrary revocation results in the minimum possible header for that revocation using the available subsets with assigned keys.

Cover Algorithm

The algorithm goes as follows:

- Construct the Steiner Tree $ST(R)$ from the complete k-ary tree using the nodes only on the paths starting from the root to the leaves corresponding to the revoked users. This is the tree induced by the set of revoked users R.

- We build the subset cover by constructing an intermediate tree T that is initially equal to the Steiner Tree $ST(R)$.
- We keep iteratively removing nodes from this tree T and their corresponding subtrees (assuming that all leaves under this subtree have been covered) and keep adding corresponding subsets to the cover by the following algorithm:
  - Find a node v_i in T such that it is the *least-common-ancestor* of leaves in T and has no internal node in its subtree. Let v_i have l children (leaves) namely $v_{j_1}, \ldots, v_{j_l}$. Let $v_{{j_1}'}, \ldots, v_{{j_l}'}$ be the children of v_i in the original k-ary tree which are ancestors of the l nodes (l can be $1, \ldots, k$) of v_i.
  - For all these leaf nodes v_{j_k} of v_i which are not the same as their ancestor nodes $v_{{j_k}'}$, add subsets $S_{{j_k}',j_k}$ to the cover. Depending on the value of l, there will be the following cases:
    * $l = 1$ which can happen if there is only one leaf left in T rooted at v_{root}, in which case, S_{root,j_1} is added to the cover. This subset has a key assigned by the Subset Difference Method.
    * $l \in \{2, \ldots, k-1\}$ which means there are l children $v_{j_1}, \ldots, v_{j_l}$ of v_i in T. In this case, $S_{i,j_1}, \ldots, S_{i,j_l}$ are added to the cover. These subsets have keys assigned by the Subset Difference Method. The set corresponding to the union of users in the other child subtrees is also added to the cover and it has a key assigned through the Complete Subtree Method.
    * $l = k$ which means there are k children $v_{j_1}, \ldots, v_{j_k}$ of v_i in T. In this case, $S_{i,j_1}, \ldots, S_{i,j_k}$ are added to the cover. These subsets have keys assigned by the Subset Difference Method. No other set needs to be added to the cover.

Maximum Header Size

There are two types of nodes in $ST(R)$: *leaf nodes* (corresponding to revoked users), and *internal revoked nodes* of degree two or more. The root node may be one of the internal revoked nodes. If not, then it is a special third kind of node of degree one. On analyzing the above algorithm carefully, we make the following claims:

Lemma 9.3.1. *A leaf node in T ($ST(R)$ to begin with) gives rise to at most one subset due to the above cover algorithm.*

Proof. If T has only one leaf (that would imply that there are only two nodes left: the leaf and the root), that would give rise to a single subset to cover all privileged users (that have not yet been covered) by the Subset Difference method. Now, if T has more than one leaf and one of them is (say) v_{j_1}, then its corresponding $v_{{j_1}'}$ in the above algorithm gives rise to one set of the cover $S_{{j_1}',j_1}$ if $v_{{j_1}'} \neq v_{j_1}$. If $v_{{j_1}'} = v_{j_1}$ then it does not give rise to any subset of the cover. □

Lemma 9.3.2. *An internal revoked node other than the root in $ST(R)$ has degree at least two. The root can have degree one.*

Proof. An internal revoked node (other than the root) in $ST(R)$ has been formed because there are at least two revoked nodes (internal or leaf) in its subtree. □

Lemma 9.3.3. *An internal revoked node of degree between $2, \ldots, k-1$ in $ST(R)$ gives rise to at most two subsets due to the above cover algorithm.*

Proof. An internal revoked node of degree between $2, \ldots, k-1$ in $ST(R)$ has k subtrees in the original complete subtree: l out of which have some revoked leaves in each and $k-l$ without any revoked leaf. At some point of time in the algorithm, when it becomes the node v_i, it will have l leaves. The union of leaves of the subtrees of this v_i that are without any revoked leaf in the original k-ary tree is one that occurs due to this internal revoked node. The other $k-l$ subtrees have one revoked leaf each in T and hence subsets occurring due to them have been taken care of by the algorithm and the subtrees get removed before this internal revoked node. These are counted as subsets generated by the $k-l$ nodes which are leaves of v_i. Now, when this node v_i becomes a leaf in T at some stage, it may give rise to another subset which has been assigned a key through the Subset Difference Method. □

Lemma 9.3.4. *An internal revoked node of degree k in $ST(R)$ gives rise to at most one subset due to the above cover algorithm.*

Proof. When such a node of degree k becomes the v_i of the above algorithm with all its k leaves revoked, all its subtrees gets deleted and it itself becomes a leaf. Upon becoming a leaf itself, it can give rise to at most one subset to be added to the cover that would have been assigned a key through the Subset Difference Method. □

Lemma 9.3.5. *An internal revoked node in $ST(R)$ gives rise to at most two subsets due to the above cover algorithm.*

Proof. Follows from lemma 9.3.3 and 9.3.4. □

Theorem 9.3.6. *The above cover algorithm gives rise to a header size of at most $3|R|-2$.*

Proof. Follows from lemma 9.3.1 and 9.3.5. □

9.3.4 Revocation and Resilience

Resilience: It can be easily observed that any user gets keys only for those subsets to which it belongs. The proof for the fact that this scheme is fully resilient follows the same lines as the security argument stated in section 3.1 based on the key-indistinguishability condition. It can be shown that each key is indistinguishable from a uniform random string for all users not in the corresponding subset.

Revocation: Any user can be revoked in the proposed scheme.

9.3.5 Summary

This scheme is an improvisation on the Ternary Tree Method needing the same header size of $3r-2$ as the Ternary Tree Method but reducing the keys per user to a certain extent. While the Ternary Tree Method uses

$$(\lceil \log_3(n) \rceil) + \frac{(\lceil \log_3(n) \rceil)(\lceil \log_3(n) \rceil + 1)}{2}$$

keys, the k-ary Tree scheme uses

$$(2^{k-1} - k)(\lceil \log_k(n) \rceil) + \frac{(\lceil \log_k(n) \rceil)(\lceil \log_k(n) \rceil + 1)}{2}.$$

Number of Users (n)	SD scheme	k=3	k=4	k=5	k=6	k=7	k=8	k=9
10^3	56	35	**35**	70	144	238	490	998
10^4	106	**54**	56	87	177	300	615	1250
10^5	154	**77**	81	124	210	363	741	1503
10^6	211	104	**95**	144	244	492	868	1757
10^7	301	135	**126**	176	259	558	996	2012
10^8	379	170	**161**	210	341	625	1125	2268
10^9	466	209	**180**	234	390	693	1255	2525

Table 9.1: Showing the number of keys to be stored by users for different values of n and k.

keys for header size same as the former. But the point to be noted here is that the number of keys does not keep decreasing with increasing k. If k is too large, the keys due to the Complete Subtree Method will grow too much in number and hence will defeat the purpose. So, there exists an optimal k for which this set of keys will be minimum in number.

Here, we consider for different values of n, the number of keys that a user needs to store for different values of k:

It is clear from the table that our scheme performs best at $k = 4$, for all practical purposes. It clearly out-performs the Subset Difference method in terms of the number of keys stored per user.

The point to be noted here is that increasing the value of k doesnot keep decreasing the required number of keys to be stored at the user. This is because of the Complete Subtree method keys that grow exponentially with increase in k.

Chapter 10

The Road Ahead

The study on Broadcast Encryption reaffirms that it is a nascent area where there is lot to explore.

The Interval Framework looked like a very promising avenue. The schemes that we have come up with in the framework, surely can be improved. Its results were similar to the PI scheme that is one of the best known scheme as per our study. In L-DAG, the challenge that lies on the road for improvement is to fit the key distribution system into some combinatorial structure (say trees) that would reduce the number of keys that need to be stored at every user. In Inverted L-DAG, the challenge is to bring in resilience into the system.

The improvisations that we have suggested in the Subset-Cover Framework through the k-ary Tree scheme, has also given us a promising insight into the problem. We have come up with and idea that combines the Complete Subtree and Subset Difference key pre-distribution methods. We have been thinking of other areas of improvement to the k-ary tree scheme. Introduction of the idea of *layering* to this k-ary tree scheme, will bring down the number of keys that need to be stored at each user. The maximum header size needs to be analyzed for the same.

Traitor Tracing is another interesting aspect of Broadcast Encryption Schemes that we have not explored during our work. Efficient Traitor Tracing Algorithms are are necessary for practical use of BE schemes. Tracing traitors for schemes developed by us would definitely be significant addition to what we have done.

Though the cover algorithm for the k-ary tree scheme has been implemented, an implementation of the key pre-distribution algorithm will also add to this work. Subset identification from the information carried in the header also has to be implemented.

In short, this looks like just the beginning of what can definitely be a significant research work that will have innumerable applications in various broadcast communiation scenarios.

Bibliography

[FN93] Amos Fiat and Moni Naor. Broadcast encryption. In Stinson [FN93], pages 480–491.

[HS02] Dani Halevy and Adi Shamir. The lsd broadcast encryption scheme. In Yung [HS02], pages 47–60.

[JHC$^+$05] Nam-Su Jho, Jung Yeon Hwang, Jung Hee Cheon, Myung-Hwan Kim, Dong Hoon Lee, and Eun Sun Yoo. One-way chain based broadcast encryption schemes. In Cramer [JHC$^+$05], pages 559–574.

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Kilian [NNL01], pages 41–62.