UNIVERSITY OF NAIROBI
SCHOOL OF COMPUTING AND INFORMATICS

Dynamic Subset Difference Revocation using One Binary Tree

AUTHOR
Austin Owino Wetoyi
P80/85241/12

SUPERVISOR
Prof William Okelo-Odongo

December 2014

This research thesis is submitted in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy in Computer Science of the University of Nairobi

Declaration

I Austin Owino Wetoyi, whose student registration number is P80/85241/12, hereby declare that this PhD Thesis entitled Dynamic Subset Difference Revocation using One Binary Tree my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning. Any uses made within it of the works of other authors in any form are properly acknowledged at the point of their use and a full list of the references employed has been included at the last page.

No part of this dissertation may be reproduced without my (the author) prior permission or the University of Nairobi

Signature:...

Date:...

This thesis has been submitted for examination with my approval as the supervisor.

Signature:...

Name: *Prof. William Okelo-Odongo*

(School of Computing and Informatics, UoN)

Date:...

Dedication

to

ochocho, chichicha and buranko,

who are yet to know what and why daddy is reading

most of, if not all, the time.

hope you all grow up to understand the content of this thesis and much more.

Acknowledgments

When I floated my title to the staff at the then Institute of Computer Science, only one person was comfortable with it and he therefore naturally ended up being my supervisor. Thanks you so much William Okelo-Odongo (woo). From our discussions, I learnt so many things that I cannot all list here but high up there is simulations/modeling.

Thanks to many people at the now School of Computing for simply recognizing me whenever I visit and finding out how I am progressing; Wagacha, Moturi, Ronge, Opiyo, Omwenga, Waema, Oboko to name just a few. That seemingly non consequential little conversation along the corridor that always ended with you wishing me well with my progress did much more than you would ever imagine.

Then there are my former colleagues at the then Institute of Computer Science, University of Nairobi when we were pursuing MSc, and former staff mates at Maseno University and Masinde Muliro University of Science and Technology some who have finished their PhDs and some still pursuing. That occasional call you make to find out how the going is does magic. Sometime it is the one that made me resume the research!

I can't fail to mention, my current staff mates at the School of Information Sciences, Moi University. Each of you believed that I can make it, I can do it. Most conversation with each of you would be littered with a question of the form "when are completing your studies?" Thank you so much for that belief.

Last but not least are my family members. You are virtually a whole clan and cannot mention each of you here. The environment I operated in was made suitable and of course sometimes unsuitable by you. Thank you so so much.

Abstract

Broadcast Encryption (BE) means transmitting information that anyone listening can access but only those selected by the transmitter, using a suitable criteria, can understand. An example is the pay TV system. An existing solution known as the Subset Difference Revocation (SDR) performs this in a *stateless* manner i.e. once the system is set up, new receivers are not allowed to join though existing receivers can each be revoked and restored as necessary.

This statelessness can lead to unnecessary high *key storage* for a receiver and *messaging overhead* due to potentially poor adjacency of receivers to each other on the *binary key tree* when the number of receivers is much smaller than the number of potential receivers.

This thesis is about a scheme that converts this static SDR into a dynamic SDR scheme. Rather than use multi-tree solution which employs multiple equally sized binary trees known as allocation units that are added and discarded on demand, it uses a single binary tree that shrinks and grows on demand.

When the positions to be assigned to active members all get filled, the tree grows by one level rather than in breadth. Similarly when the number of members is not more than half the tree capacity, the tree shrinks by one level. Therefore, there is no need to know the maximum number of potential receivers in advance, a value that can be difficult to estimate in practice.

In this thesis, we investigated how this solution compares in efficiency to the multi-tree solution that uses allocation units in terms of *key storage* at the receiver, the *multicast* cost and the inevitable *unicast* cost. The methodology used is simulation.

The results obtained show that the single-tree solution in typical usage performs, at worst, like the multi-tree solution and this is the major contribution.

Keywords: dynamic scheme, broadcast encryption, stateless scheme, binary tree, key tree.

List of Abbreviations and Acronyms

BE	Broadcast Encryption
CAS	Conditional Access System
CS	Computer Science
CSM	Complete Subtree Method
CW	Control Word
DOE	Design of Experiment
DRM	Digital Rights Management
DVB	*Digital Video Broadcasting*
ECM	*Entitlement Control Message*
EMM	Entitlement Management Message
KEK	Key Encryption Key
LKH	Logical Key Hierarchy
PDK	Personal Distribution Key
SDS	Subset Difference Scheme
SK	Service Key
TEK	Traffic Encryption Key
TSC	Theoretical Computer Science

Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Background

Broadcasting means transmitting information through a medium or channel that is accessible to more than one receiver. Radio transmission for example uses air as a medium and anyone who has a radio receiver is able to listen to the broadcast. Most of the time, broadcast communication is one-way[1].

There are many situations where one or more senders want to transmit a message to only a selected subset of receivers, using a one-way *broadcast channel* - where not everyone that has access to the channel should have access to the content passing through the channel. These are membership-based applications, such as pay-per-view and specialized information services (e.g., stock price, live news) that require that information content be delivered to, and only to, subscribed members or authorized receivers.

This should be achievable using *encryption* - since the message must be protected from other receivers than the ones that have been selected to receive it, for example by paying for it, it can be *encrypted* before it is sent. This is what is called *broadcast encryption*. Broadcast encryption is a problem because partly, in practice, receivers of the message to be sent do not all share the same *decryption key*, therefore multiple copies of the message must first be *encrypted* separately for these users before sending each encrypted copy of the message. Some equivalent definitions from literature of the term *broadcast encryption* are:

- The cryptographic method for a centre to efficiently broadcast encrypted digital content to a system of users so that only an intended subset can correctly decrypt it (Bhattacherjee & Sarkar, June, 2012).
- Broadcast encryption is an application of cryptography which allows one to broadcast a secret to a changing group of intended recipients in such a way that no one outside this group can view the secret (Obied, April 2005)
- A method to efficiently broadcast information to a dynamically changing group of users who are allowed to receive the data (Naor, Naor, & Jeff, 2001).
- Broadcast encryption is a way to broadcast information securely, that is to say, broadcasting a *secret* to a dynamically changing set of intended

[1] receivers cannot send anything back to the broadcaster.

recipients in such a way that no one outside this set can recover the secret (Obied, April 2005).

- Broadcasting a message (e.g. a key to decipher a video clip) to a dynamically changing privileged subset of users in such a way that non-members of the privileged class cannot learn the message. (Fiat & Naor, 1994).
- Broadcasting the same message to all users, and those users in the privileged group recover the message while all others derive nonsense or nothing at all (Lassalle, January 2005).
- *Broadcast encryption* is the *cryptographic* problem of encrypting broadcast content (e.g. TV programs) in such a way that only qualified users (e.g. subscribers who've paid their fees) can decrypt the content. (Wikipedia, 2014).

Like general broadcast, in broadcast Encryption, two-way communication is not allowed on the channel and other channels may not be used except when setting up the system and distributing initial secret decryption keys (or key material) to possible receivers.

As we see in the following sections, *broadcast encryption* is a problem and a solution to the problem is called a *broadcast encryption scheme*. In membership-based applications, it is also desirable to be able to revoke users who have become untrusted without effecting the remaining members. A *Broadcast encryption scheme* implements this by revoking the decryption keys of such users, thus, *broadcast encryption scheme* are also known as *key revocation schemes*. So an efficient *broadcast encryption solution,* or *scheme* as they are more popularly known, would make it possible to revoke departing users and send further communication only to the remaining users. Similarly the scheme would make it possible to add new members into the set of privileged users. Naturally, the non-members are curious about the contents of the message that is being broadcast, and may try to learn it. The schemes should be resilient to any subset of revoked users that collude as well as any disjoint subsets (of any size) of privileged users. The scheme is considered broken if a user that does not belong to the privileged set can read the transmission.

There are different opinions about when the idea of broadcast encryption was introduced. For a little detail on this, see (Anderson, 2005)..

1.2 The Broadcast Encryption Problem

In traditional cryptography, the focus is to enable secure communication over an insecure medium. This is accomplished by *symmetric key* (also known as *secret key*) cryptography. In *symmetric key cryptography*, the key used for encryption is the same key used for decryption, thus anyone having the key can both encrypt and decrypt messages. Normally, the scenario where only two parties are communicating (and thus share the key) is considered. Symmetric key cryptography works well for a group communication too and is also the one used by Broadcast Encryption schemes.

However this works correctly only as long as the group is static (i.e. no one leaves or joins). Should someone leave, they must be prevented from being able to decrypt further group communication. However, the only means the group has of communicating securely is by using the shared group key, which the party now excluded also knows! The challenge arises from the requirement that un-subscription of some users should not affect the remaining users. This is the *Broadcast Encryption Problem*. The solution to the problem, as already pointed out, is known as *Broadcast Encryption Scheme* or *Key Revocation Scheme*.

The *Broadcast Encryption Problem* consists of two parts (Anderson, 2005)..

- Deciding that a particular user should have particular decryption key(s).
- Selecting the encryption keys to use when sending a message to specific subset(s) of users.

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

Typically broadcast encryption scheme allows licensors to "revoke" individual users, or more specifically, the decryption keys associated with the users. Thus, if a given user's keys are compromised and published, the licensors can simply revoke those keys in future content, making the keys useless for decrypting new broadcasts.

The problem of rogue users sharing their *decryption keys* with unqualified users is *mathematically insoluble*. There are algorithms known as *traitor tracing algorithms* that aim to minimize the damage by retroactively identifying the user or users who leaked their keys, so that punitive measures, legal or otherwise, may be undertaken. In practice, pay TV systems often employ *set-top boxes* with *tamper-resistant smart cards* that impose physical restraints on a user learning their own decryption keys.

1.3 How Broadcast Encryption Schemes Work

In all *Broadcast encryption Schemes* there is a pre-processing phase in which the centre distributes a number of *decryption* keys to the users. It is these keys that the centre later uses to encrypt the messages to be sent to users. The obvious and simple solution is to distribute one unique *symmetric key* to each user, for a total of n keys. The broadcast centre then encrypts the message once for each privileged user with the key of the user and finally broadcasts all these encrypted messages over the broadcast channel. This makes the space requirements for the users be $O(1)$[2], but the transmission length is $O(n)$[3] where n is the number of receivers! This means that the *message expansion*, the number of transmissions per message, is equal to the number of privileged users. Clearly, this scheme is only useful if the number of privileged users is small.

[2] Each user only needs to store one key so it is very efficient in terms of storage at users.

[3] This requires a very long transmission (the number of members $\times$ the length of the message)

To reduce the *message expansion*, broadcast encryption schemes group users into sets and distribute keys giving all users in the same set a common decryption key. This way, the number of broadcast transmission reduces to the number of sets whose members are in the privileged set P. But more often than not, a message to be broadcasted is meant for users in more than one set and therefore still requires transmission by *broadcast encryption* i.e. the message is encrypted multiple times separately with the key of each set so that the message is decryptable by those users in the set. So it is practically impossible to reduce the message expansion to 1.

1.3.1 Traffic Encryption Keys and Key Encryption Keys

Unless the broadcaster is sending the message to all the users – using one key that they all share - it would clearly be a bad solution to transmit the actual message using *broadcast encryption* as the message may be for example a video stream of a movie which will be quite long.

Think of this; to send an encrypted message such as a video stream to *k* subsets of users, a broadcaster would first have to encrypt the video with the enciphering key assigned to each subset and transmit the *k* versions of the video. A cable operator would typically be providing services to a million users!

Instead the actual message is encrypted just once using one common key, known as *Traffic Encryption Key* (TEK)[4] and broadcasted. Obviously for the legitimate users to recover the message, they must have this key. It is this TEK that is transmitted to them using broadcast encryption. How? There is a preprocessing phase in which the center, not knowing the privileged set of users nor the common key (TEK), distributes a number of keys to the users. It is these keys that the centre uses to encrypt the TEK. For every set of users who should receive the TEK, the centre encrypts the TEK with the key assigned to that set. These keys assigned to the user sets are thus a kind of *Key Encryption Keys* (KEKs). There are two approaches to how to deliver these multiple encryptions of the TEK.

[4] Also known as *Group Key* or *Media Key* or *Session Key* or *Common Group Key*. The most popular in literature of these seem to be "*session key*".

In the schemes based on *Key Management Block*, the broadcaster composes a message with two parts - the *message body* that contains the protected actual secret message and the *message header* that consists of the multiple encryptions of the TEK. Once a legitimate receiver has recovered the TEK, they go ahead to use it to recover the secret from the message body. A *key management block* is a block of data located at the beginning of a broadcast or pre-recorded onto some type of blank media, most often a smart card. From this key management block, each recipient can derive the *management key* (KEK). A device not in the privileged group of devices even with access to the encoded data will derive the wrong answer from the key management block. Restricted devices can attempt to process the key management block but they will not yield the correct key. (Lassalle, January 2005)..

In the *Key Pre-distribution* based schemes, the broadcaster B broadcasts each encryption of the TEK. But this really is a multicast of each encryption by B because only intended receivers will be able to decrypt their copy. B could as well package all the encryptions into one message. Once the centre has transmitted the TEK, it can start broadcasting the messages encrypted using the *session key* (TEK).

The message header in the *Key Management Block* scheme must be in a suitable representation such that members can compute what part of the message to decrypt using one of the keys they are assigned at initialization. This is also true for *Key Pre-distribution* if all the encryptions of TEK are packaged into one message.

Thus, the TEK is encrypted using *Key Encryption Keys* (KEKs) and then multicast by the key server. This TEK is shared by all *privileged users*[5] no matter which subset they belong to. When a member joins the group, the TEK must be changed to ensure that the newly joining member cannot decrypt previous communications (a requirement known as "*backward confidentiality*"). Similarly, the TEK must be changed when a member leaves the group to ensure that future messages cannot be decrypted by the departing member (a requirement known as "*forward confidentiality*"). In addition, the TEK could also be updated at timed intervals.

[5]Also known simply as *members*. Other terms one may come across are *intended recipients, active users* and *receivers* among others.

1.3.2 Rekeying

The procedure of delivering a new TEK to members is known as *rekeying* - basically a *multicast* by the key server. During rekeying, a user receives this key only if the user belongs to one of the sets the key server is sending it to. In other words the user recovers the TEK if, in the user's set of keys, the user has one that can decrypt the message containing the TEK. Even if they know all the broadcast messages of the other users, a coalition of non-privileged users cannot recover any information regarding the common key.

Whichever of the two approaches is used to deliver the TEK, the rekeying message is known as the message header. The transmission overhead of a scheme is determined by the *header length* (the number of encryptions of the session key). The header in a broadcast message is the most important part when one analyses any broadcast encryption scheme.

To summarize, a basic broadcast encryption scheme consists of four algorithms which can be performed in *polynomial time*[6] as illustrated in Figure 1. These algorithms are as follows:

- Initialization:

  This is the preprocessing/initialization phase in which the key distribution centre, G not knowing the members of P nor the common key (TEK), generates[7] the decryption keys and selects sets of *users* and distribute the keys giving all *users* in the same set a common decryption key. Thus a user gets a set of keys one for each of the sets they are a member of.
- Registration

  The broadcaster, B identifies the sets of users who should receive the message. This algorithm is used to register new users that can view some secret message $s \in S$. In particular, whenever a new user $u \in R$ wants to join P, then the key server, G removes u from R and adds it to P. If a user $u \in P$ wants to leave P, then G removes u from P and adds it to R.
- Broadcast Encryption Phase

  For every set registered by the previous algorithm, the broadcaster B, encrypts the message once with the key assigned to such a set for transmission.

---

[6] The time taken or number of operations performed is a polynomial function of n, the size of the problem/input i.e. $T(n) = O(n^k)$ for some constant k.

[7] This can also be done by a trusted authority

- Decryption:

  This is done by the users. Each user tries to decrypt each message using each of the keys in their key ring.

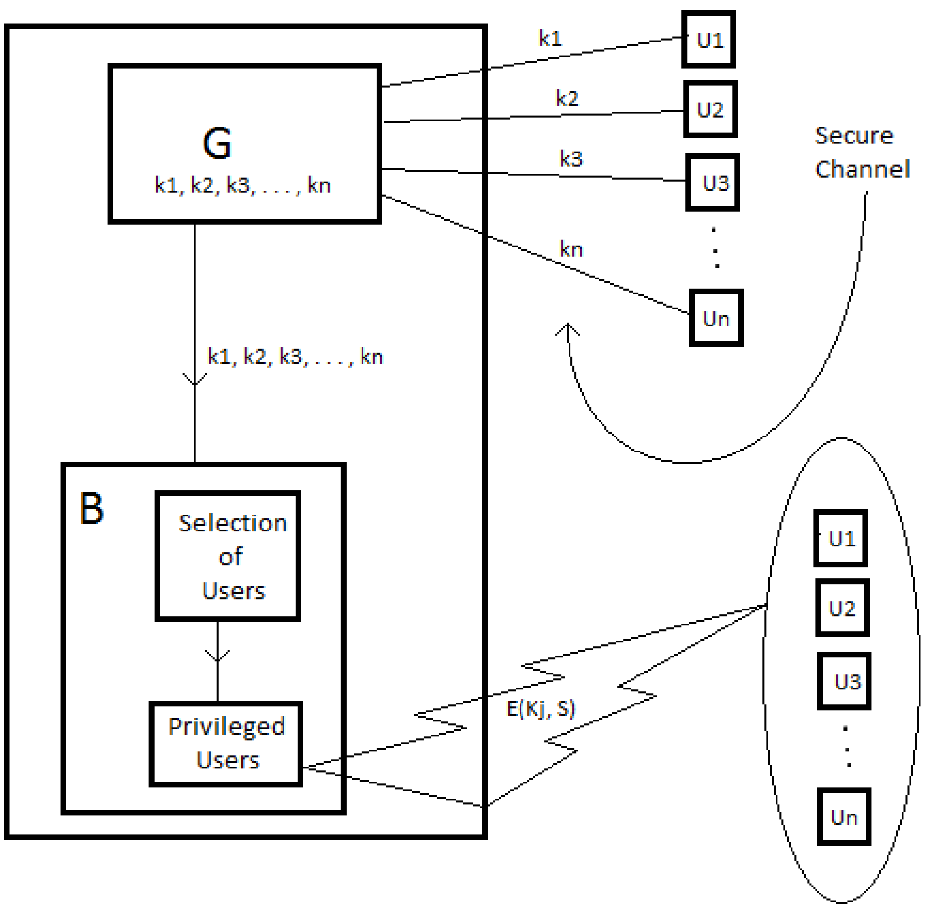


Figure 1: The Rekeying Process

1.4 Importance of Broadcast Encryption

Broadcast encryption schemes provide many benefits over other technologies especially when used in the realm of content protection. Copyright protection using *Digital Rights Management* [DRM] techniques is an important application of Broadcast Encryption. Out of the different facets of copyright protection, Broadcast Encryption handles the content protection part. The application of Broadcast Encryption systems is pretty wide in the implementation of DRM for content protection in digital data distribution technologies such as pay-TV, Internet or mobile video broadcast, optical discs, etc. It can be useful in pay-TV system distributing copyrighted information of CD and DVD disks, and multicasting music and video on the internet. An alternative possible solution is public-key cryptography. The advantages of Broadcast Encryption over public-key cryptography (Lotspiech, Nusser, & Pestoni, August 2002) stem from the following:

- *Its low overhead*

  Broadcast encryption is fast. All its calculations are done using simple symmetric encryptions. In contrast, actual public key calculations require exponentiation operations over a finite field. The processor load to calculate a management key in a broadcast encryption scheme literally requires less than 1,000 times the load required to perform a public key signature calculation.

- *Its Revocability*

  The ability to remove compromised keys from the system is a major advantage to provide longer life and durability to the system. Without a means to revoke compromised individual keys, a public key system degenerates into a global/shared secret scheme: The first break (one key is discovered by an unauthorized party) defeats the entire system. If a proposed public key system contains a flaw—and, sadly, many do these days—it is almost axiomatic that the revocation information fails to travel through the system.

- *Its resistance to reverse engineering*

  Public-key systems perform a handshake at the link-level requiring keys to be placed in the link-level code where they might be easier to find by malicious users. On the other hand, since their systems are one-way, broadcast encryption schemes have the advantage that they can hide their keys much deeper in the software making the keys more difficult for malicious users to discover.

These three advantages of Broadcast Encryption are of utmost importance in consumer electronics. **Advanced Access Content System (AACS), Blu-ray** and **HD DVD** use *broadcast encryption* schemes.

1.5 Evaluation parameters of Broadcast Encryption schemes

The simple broadcast scheme described at the start of section 1.3 makes n transmissions to send one secret message when sending the message to n users because it has to encrypt each transmission with each user's unique key. This results in minimal storage on the user's side since each user only needs to store one key. But it also results in the longest possible transmission since the message must be sent to each member encrypted separately with that user's key.

The other extreme is instead to distribute one key to each possible subset of users, yielding a total of $2^n - 1$ keys. That is, one key for each subset except when no users are in the privileged set and therefore there is no need to transmit at all. This means every user receives the keys corresponding to the subsets they belong to thus each user then needs to store one key for each subset he belongs to, in total 2^{n-1} keys per user! This scheme yields no message expansion because whichever subset is the privileged one, there is always one key corresponding to that particular subset.

These two schemes might be appropriate in certain specific scenarios with few users and where one condition is extremely important while the other is totally irrelevant. It is, however, highly likely that these scenarios are not very common and instead there are different conditions of varying importance but none can be completely ignored.

Generally, efficient broadcast encryption schemes must be efficient in both measures, i.e. *transmission length* and *storage at the user's end*. In practice, several solutions exist offering various tradeoffs between the increase in the size of the broadcast, the number of keys that each user needs to store, and the feasibility of an unqualified user or a collusion of unqualified users being able to decrypt the content. They take into consideration many other factors some of which are described ahead in within this section. The list is not exhaustive and it deals almost exclusively with performance, not so much with security. The reason for this is an assumption that the encryption algorithms are strong enough for the purposes they are used for (Anderson, 2005). Some of these parameters are used to evaluate the single-tree solution proposed in this thesis.

1.5.1 Collusion resistance

The collusion resistance is a measure of the security of a scheme, how well it resists attacks from cooperating non-privileged users. If a scheme with perfect collusion resistance is used then even if all non-privileged users cooperate they will not be able to decrypt the message.

1.5.2 Backward secrecy/confidentiality

Backward secrecy is the property that any newly authorized users should not be able to decrypt messages that were sent before they were privileged users. In other words backward secrecy means that a privileged receiver cannot use the information it has to recover material that was broadcast before it was added. Imagine a situation when the broadcaster, B wants to broadcast secrets $S = \{s_w\}$ where such secrets messages are related, that is, one can think of it as a TV show which has k minutes and each s_i corresponds to 1 minute in the show. If a revoked receiver $u \subset R$ eavesdrops and records secrets $s_1, s_2, \cdots, s_{k-1}$ and right before s_k is broadcast u registers, that is, u gets removed from R and added to P. If B now broadcasts the ciphertext of s_k, and u uses his knowledge of recovering s_k to recover $s_1, s_2, \cdots, s_{k-1}$ and fails then backward secrecy is maintained.

1.5.3 Forward secrecy/confidentiality

Forward secrecy means that a revoked user should not be able to decrypt any future messages sent to the privileged set. In other words, forward secrecy means that when a privileged receiver is removed from P then it must not be able to continue viewing protected content of the broadcast. Imagine a situation similar to the one described above, that is, a broadcaster B wants to broadcast secrets messages $S = \{s_1, s_2, \cdots, s_k\}$ where such secrets are related. If a privileged receiver $u \in P$ receives $s_1, s_2, \cdots, s_{k-1}$ and right before s_k is broadcast u leaves P, that is, u gets removed from P and added to R. If B now broadcasts the ciphertext of s_k, and u uses his knowledge of recovering $s_1, s_2, \cdots, s_{k-1}$ to recover s_k and fails then forward secrecy is maintained.

1.5.4 Amount of keys to store at receiver

The receivers can be small devices with a limited possibility for storage, especially secure storage, and thus the amount of keys that each receiver is required to store is an important parameter in many applications.

1.5.5 Amount of keys to store at sender

Although the sender usually is considered to have far more storage available than the receivers, there is of course some limit even on the amount of keys that the sender can store.

1.5.6 Amount of heavy computations required of receivers

As an alternative to storing a large amount of keys some schemes require users to store only a little key material but instead perform many computationally heavy operations on this material. Depending on the scenario, this can be a good solution or not. If the receivers have very limited computational power then it is important that they are not required to perform many demanding operations. In general there are no such restrictions on the sender which is assumed to be quite powerful, although of course extreme cases can occur where a scheme involves computations that are too heavy for the sender.

1.5.7 Number of broadcast transmissions per message

An important parameter is the number of broadcast transmissions of the same message, each encrypted with a different key, that have to be made for each message. This is also known as message expansion.

1.6 Statement of the problem

In stateless schemes, several secret decryption keys are distributed to the users when they join the system and these keys are never updated by the scheme. This implies that there is never any updating of secret keys, the keys given to the users at setup are the keys that are used throughout the lifetime of the system. We term such users stateless and the scheme a stateless scheme. It is the disadvantages that trace their root to the stateless of the Subset Difference Revocation Scheme that this thesis addresses - by converting the *stateless Subset Difference Revocation Scheme* to a *stateful*. Because the solution allows new users to be admitted, the solution can also be called *Dynamic Subset Difference Revocation Scheme* (see the title of this thesis). Note that *stateless Subset Difference Revocation Scheme* is also known as a *Static Subset Difference Revocation Scheme*.

The following are the shortcomings of stateless schemes in general. The cause of these shortcomings is easily visible as the statelessness of the schemes.

1.6.1 Static

The key server initially generates a key tree large enough to accommodate the currently active members and no more members are allowed once the initial set up has been done. The keys used to encrypt the session key are never changed neither is the secret information, I_u given to each member as part of the initialization. Why not just generate new secret information and hand to a new user? This is not possible by design. The secret information is determined by the position of the member on the key tree. As a consequence, once a leaf position in the key tree is assigned to a member x, that position cannot be assigned to any other member even when x is currently not in the group. Typically, in static SDR, a returned member (a member joining the group again after leaving) is assigned to the position that the member was assigned last time.

1.6.2 Large Key Tree

The key server in static SDR needs to maintain a key tree large enough for all the potential members, *N*. This is a consequence of the previous property. When users are revoked, the leaves they were assigned to (positions they occupied) on the key tree cannot be assigned to anybody else. So what happens to the size of the Key Tree as active users reduce? Nothing!

1.6.3 Message Expansion

When the session key (one message) is to be sent to m subsets of users, the number of broadcast transmissions is also m, each encrypted with a different key, that have to be made for each message. This can also be referred to as *message overhead*, *message expansion* or even *time per message* as the number of broadcasts that have to be made in order to send one message will affect the total time to send the message. In some cases this parameter can be referred to as *cover size*.

This number of the resultant subsets m is determined by the adjacency (or positions) of members of m in the key tree of size n. Generally speaking, under the assumption that member activity is independent of position in the key tree, the larger the difference between n and m, the more likely that active members are sparsely distributed in the key tree, resulting in $O(m)$disjoint subsets. On the other hand, the smaller the difference between n and m, the more adjacent the positions of active members in the key tree.

1.7 The proposed single-tree solution

As stated in $section\ 1.3$, all *broadcast encryption schemes* consists of a pre-processing phase in which the centre distributes a number of *decryption* keys to the users that the centre uses later to encrypt the messages to be sent to users. In *stateless schemes* these keys don't change throughout the lifetime of the system. Some of the disadvantages are that the centre must know in advance the number of users, and more users cannot be admitted once the system is up and running.

This thesis is about converting the well-known stateless *Subset Difference Scheme* into a *statefull scheme*. This involves the users being given one special lifetime secret key that the centre uses to transmit new *broadcast encryption keys* to each new user or a returning user who was previously revoked but whose keys must be changed – a *unicast*.

In literature, there seem to be only one attempt to do this (Chen, Ge, Zhang, Kurose, & Towsley, 2004). The solution proposes using fixed-equally-sized binary trees that are added and removed on demand. All the trees share a virtual root, therefore the tree as a whole increases in breadth only.

The solution in this thesis proposes uses a single binary tree that increases or reduces in size on demand. This is because the goal is to improve the stateless SDS which uses a key tree. More details on this are presented in section 2.1. Because the tree is both complete and full – a perfect binary tree (Black, 2014)- it increases in height, and also breadth to accommodate more users and similarly decreases in both height and breadth when users reduce in number.

This implementation addresses each of the three shortcomings of the stateless schemes discussed in *section* 1.7:

- Static Nature

  Each user (in this proposal) is given a special secret key that is used to transmit the secret information I_u. Each time a new user is admitted, they are sent I_u encrypted with this special key. Also when a user who was previously revoked is reinstated, the user must be unicast their I_u if the tree they are returning to is not the one they were in when they were revoked – like the user trying to occupy a different location on the key tree, therefore their I_u is different.
- Large Key Tree

  The tree is always only big enough to accommodate the number of privileged users – members.
- Message Expansion

  The users are always packed as close as possible to each other, therefor they have common ancestors in the key tree. A node corresponds to the encryption key to use when encrypting a message for users who are leaves in the subtree whose root this node is. So when users are packed close together, chances that they share a common key are high and therefore message expansion is low.

1.8 The Research Question

The broadcast encryption problem is a big problem which has been studied by many authors. The scope of this thesis is only a small aspect of this problem. The specific problem this thesis addresses is the static nature of Subset Difference Schemes.

Thus *the research question* for this thesis is:

In a situation where users can change their state, can a single tree that grows and shrinks, according to the population size of members, efficiently implement a stateful revocation scheme in terms of the following parameters, in comparison to the multi-tree solution proposed in (Chen, Ge, Zhang, Kurose, & Towsley, 2004)?

- The *key storage cost at the member side,*
- The *multicast cost* and
- The *unicast cost.*

1.9 Research Objectives

The parameters outlined in the research question are measurable quantities. The research objectives pursued in order to answer the research questions are:

- To determines the factors that influence these BE evaluation parameters specified in the research question
- To develop a simulator program for each of the two solution whose output variables are each of a value of the evaluation parameters specified in the research question.
- To use the simulator to investigate the performance of the single-tree solution and the multi-tree solution in terms of the *key storage cost at the member side*, the *multicast cost* and the *unicast cost* under various conditions.

1.10 Significance of the study

Conditional Access System (CAS) is used for securing the digital TV content. It's one of the most important part of Pay-TV system. There are several practical CAS systems now. In *Digital Video Broadcasting* (DVB) system, for instance, a typical CAS system usually include a three-level encryption scheme.

- The raw content is scrambled or encrypted by *control word* (CW). The CW corresponds to the *Session Key* in Broadcast Encryption Schemes.
- The CW is encrypted by the *Service Key* (SK) and embedded into *Entitlement Control Message* (ECM). The *Service Key* corresponds to the *broadcast encryption key* in Broadcast Encryption Schemes. In Broadcast Encryption Schemes it is a referred to as *key encryption key* (KEK) and *Entitlement Control Message* correspond to the *message header*.
- The SK is encrypted by Personal Distribution Key (PDK) of authorized users and embedded into Entitlement Management Message (EMM). The PDK correspond to the special secret key referred to in Section 1.6. In stateless schemes the KEK are permanent but in the solution proposed in this thesis each user has state and therefore must receive a new set of KEKs for each subset they belong to. This message that contains the KEKs is a unicast message to the individual. In DVB system, this message is known as EMM and the secret key used to encrypt it is the PDK. PDK is a fixed information only known to the service provider, and is embedded into the user's secure module (smart card)

Scrambled content, ECM and EMM are broadcasted to public. The refresh frequency of CW is about several seconds or several minutes, and renew interval of SK is usually from several hours to several days. The refresh rate of CW and SK depends on several conditions such as system capacity, system security evaluation, network performance and so on. PDK is a fixed information only known to service provider, and is embedded into user's secure module (smart card). At user side, each receiver first filters the corresponding EMM messages and decrypts the SK, and then decrypts ECM using SK. After authorized user gets CW from ECM, he could descramble the content.

However, this hierarchical encryption scheme is not efficient for frequent key refreshment. If there is a CAS serving M subscribers and N channels, then key distribution scheme needs to generate N ECM messages for channel's CW refreshment, and $N \times M$ EMM messages for service key refreshment. (Zhang, Yang, Liu, & Tian, May 22-24, 2009).

There is a case to show how the traditional CAS system suffers from key refreshment. Suppose a CAS system contains one million subscribers and 30 channels. The control word for scrambling is refreshed every ten seconds. The service key for ECM generation and CW encryption is changed every day. The minimum bit-length of ECM message is about 168 bits. The bit-length of EMM message is at least about 488 bits. Then the minimum bandwidth for ECM transmission is:
$(1 \times 30) \times 168 / 10 = 504\ bps$

In order to improve subscribers' user experience, the CA system has to broadcast EMM repeatedly. Suppose the CAS system needs to ensure every subscriber is receiving renewed EMM every hour, then the required bandwidth for EMM broadcast is:
$(1 \times 10^6 \times 30) \times 488 / 3600 \approx 4.07\ Mbps$

From the case study above, we could find that EMM message broadcast needs too much bandwidth for service running, and key refresh problem limits the CAS system for mass-scale environment such as *Direct-to-Home Broadcasting* (DTH).

In order to overcome the EMM refreshment problem, the authors of (Zhang, Yang, Liu, & Tian, May 22-24, 2009) shows how we can introduce the broadcast encryption scheme into CAS system and give an analysis of its advantages and challenges.

But their proposed solution is still based on a stateless scheme. The broadcast centre decides the CAS system's capacity, and then make a broadcast encryption system according to capacity and security parameters. In order to finish the setup phase, the centre has to pre-assign the subset keys for every user ID in the broadcast encryption scheme and embeds them into the smart-cards in manufacture or burn-in phase. And the mechanism for informing which subsets the smart-card belongs to is also required.

With their solution therefore, we could still end up in a situation where rekeying cost is unnecessarily high because users are scattered at the base of the key tree but one cannot shift them to reduce this cost. Also the solution breaks down if the system's capacity is to be exceeded.

The single-tree solution has all the advantages discussed in (Zhang, Yang, Liu, & Tian, May 22-24, 2009) plus the advantages of being able to change the state of a user i.e. assign them fresh KEKs instead of permanent pre-assigned keys. There is no possibility of exhausting the system capacity and the rekeying cost is guaranteed to be lower. The price to pay for this, as we later, is the occasional unicast to each member when the new set of KEKs has to be sent to each member.

1.11 Convention, Terminology and Notation

Broadcast encryption involves a network denoted as Q made up of $n+2$ nodes; a set $U = \{u_n\}$ of n users (receivers) and two other nodes the *broadcaster* or *broadcast centre* denoted as B (a.k.a the transmitter), and the *key server* or simply the *server* denoted as G. The broadcaster B has a set of m secrets $S = \{s_m\}$ which can only be viewed by a set of *privileged users* P[8] where $P \subseteq U$ and the key server G has a set $K = \{k_w\}$ of w *symmetric keys*. Any key $k \in K$ can be used to encrypt a secret $s \in S$ using an encryption function $E(k,s)$ and decrypt s using a decryption function $D(k,s)$. G is responsible for both generating the *symmetric keys* and associating them with the users in P. The set of *non-privileged*[9] *users* (users who will not be able to decrypt the message) will be denoted as R where $R \subseteq U$ and $R \cap P = \emptyset$ along with $R \cup P = U$ must both be true. Any user $x \in P$ is said to be a *privileged user* and any user $y \in R$ is said to be a *revoked user*. If a user $x \in U$ and $x \notin P$ then x is considered an *eavesdropper* and it is *computationally infeasible* for x to recover s.

In *symmetric encryption* techniques, the key used by B to encrypt the secret s is also the key used by $x \in P$ to recover s. The key is known as a *shared key* since the sender and the recipient must both use it. The key must be known only to members of the group using the key and therefore this key is also known as the *secret key* and the technique as *secret key cryptography*. Broadcast encryption schemes use symmetric encryption techniques.

[8] Some authors use the symbol M for "members" instead of P but P is the standard notation. These are the users who will be able to decrypt the message.

[9] While the term privileged set is a common one in the broadcast encryption literature the name for the non-privileged set varies between different authors. Many use the term revoked set, thus the notation R, but someone may feel that this term is not quite suitable because in order to be revoked, a user must first have some right, be authorized to something, but a non-privileged user may very well never have been a privileged user and thus has never been revoked. The R notation for the non-privileged set, however, can be considered a standard notation.

It is assumed in this model that there is a "secure" key server which uses a protocol to authenticate the *broadcaster*, and the *users* (e.g. has a list of privileged and revoked users). Furthermore, it is assumed that a communication medium exists between the nodes in Q and there is a secure channel between G and B.

The broadcast encryption scheme proposed in thesis and the others it is based on and discussed in Chapter 2, Literature Review, use a binary tree to organize users and keys. In Table 1 are some of the terms and definitions about binary trees that the reader should be familiar with to be able to understand most of the rest of the thesis. Each of these definition are from (Black, 2014) with slight modifications where necessary for clarity for the purpose of the reader of this thesis.

Table 1: Some tree terminology

Term	**Means...**
Node	A unit of reference in a data structure. Also called a vertex in graphs and trees.
Root node	The distinguished initial or fundamental node of a tree.
Parent of a node	The node conceptually above or closer to the root than the node and which has a link to the node. The root is the only node with no parent.
Child of a node	Any node it has a link to and is one level further from the root. Every node, except the root, is the child of some other node.
Binary tree	An empty structure or a node known as the root node which has, as its children, the roots of two disjointed binary trees, known respectively its left subtree and its right subtree. (A tree with at most two children for each node.)
Size of a tree	The number of nodes of the tree
Edge	A connection between two vertices (nodes) of a graph. In a directed graph (rooted tree), an edge goes from one vertex, the source, to another, the target, and hence makes connection in only one direction. Also known as arc.

Term	**Means...**
Path (between two nodes)	A sequence/list of nodes where each node has an edge from it to the next node in which the two nodes are the terminal nodes of the sequence.
Path length	The number of edges in the path, equivalently, one less the path. A path with one node does not contain an edge, therefore its length is 0.
Depth of a node	The path length of the path from the node to the root.
Leaf	A node in a tree without any children. Also known as external node or terminal node.
Height of a tree	The maximum depth of any leaf from the root. If a tree has only one node (the root), the height is zero. The height of an empty tree is not defined. The height of a tree is also known as the order.
Level	Any depth in a tree that is not empty ie that contains one or more nodes.
Subtree of a node	A tree whose root is a child of the node.
ancestor of a node	All the nodes on the path between the root and the node excluding the node itself. The root is the only node which has no parent and therefore ancestor.
Descendant of a node	Each node that lies on the path from the node to a leaf. The leaf is the only item which has no child and therefore descendant.
Internal node	A node of a tree that has one or more child nodes, equivalently, one that is not a leaf. Also known as nonterminal node.
Siblings	Nodes that have the same parent.
Degree (of a node)	The number of child nodes the node has.

Chapter 2: Literature Review

2.1 Broadcast Encryption Schemes

A broadcasting scheme involves encrypting a message so that more than one privileged receiver can decrypt it. To achieve this, privileged receivers are grouped together either dynamically or statically according to the scheme being used.

The performance of any broadcast encryption scheme depends on how the privileged receivers are grouped. All modern Broadcast Encryption schemes use a graph to organize keys and users. The graph is called a key graph and can have varying properties but the most common type of graph discussed in the literature seems to be a *directed acyclic graph* which forms a *rooted tree* with some maximum degree d, a *key tree*, and that is the only type considered in this thesis. It is, however, worth noting that a key graph in general can be any directed acyclic graph (Anderson, 2005)..

In particular, the tree considered in this thesis is a perfect binary tree – a full and complete tree (Black, 2014) of degree 2. All the nodes of the tree (see Figure 2) are k-nodes to signify that they store or correspond to keys. Each of the leaves of the key tree is associated with a u-node which is a node that stores or corresponds to a user. In a diagram, the u-nodes have no incoming edges. See Figure 2. A directed path in the graph from a u-node u to a k-node k represents the fact that user u has key k.

In a key tree, the users are organized in a hierarchical fashion (see Figure 2) so that all users with a common ancestor also have a common key. Thus these related users can form their own group and broadcast to all users in a group is done using their common key – the key stored in their common ancestor.

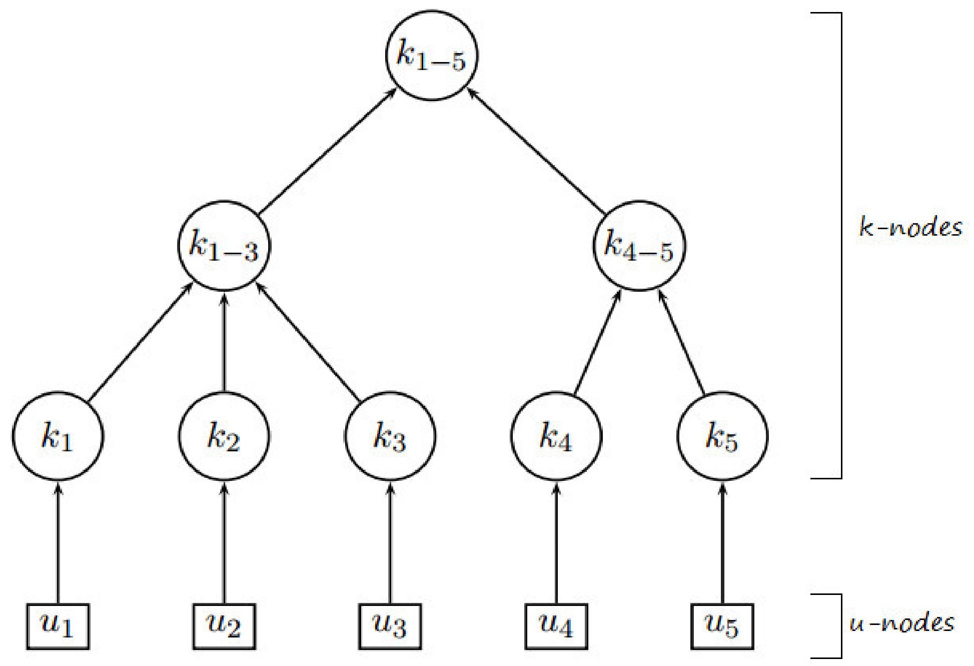


Figure 2: An example of a key tree showing users and the keys.

In Figure 2, all users $\overline{u_i}$ have their own key $\overline{k_j}$. Users $\overline{u_1, u_2 \text{ and } u_3}$ have the common key $\overline{k_{2-3}}$ and user $\overline{u_4}$ and $\overline{u_5}$ have the common key $\overline{k_{4-5}}$. All users have the common key $\overline{k_{1-5}}$.

At initialization phase, the key server S generates random keys and assigns them to each node in the tree.

As already pointed out, Broadcast Encryption scheme begins with an initialization phase where every user is given a set of secrets decryption keys. The *sender* (or a trusted authority) initially generates several secret decryption *keys,* selecting subsets of *users* and distribute the keys giving all *users* in the same subset a common decryption key. Thus a user gets a set of keys one for each of the subsets they are a member of.

In actual practice, the set of keys a user receives from the sender is really some key information I_u from which keys can be derived i.e. what the user is given is not a set of keys as implied above but some key information I_u from which the user can derive each key associated with all possible subsets to which they may belong. The size of I_u is called the *user storage*. Schemes where I_u (and the keys it is used to derive) is never updated are called *stateless*, whereas those where it is updated are called *stateful*.

A stateful broadcast encryption scheme requires that all the receivers have to be able to update the stored keys, usually when receivers are added or removed from the privileged receiver set P. This implies that any receiver $r \in P$ must be connected all the time to the broadcast network Q in order not to lose any key update message that might be sent.

2.2 Logical Key Hierarchy – A Stateful Scheme

The *Logical Key Hierarchy* (LKH)[10]scheme is an example of a stateful broadcast scheme. All users in the key graph of LKH scheme are privileged users (in P) and the non-privileged users (in R) are not in the key graph. This means that when broadcasting to P the root key can be used as all privileged users have this key and no other user has it. Every user on the tree must know its own key and the keys in the path from its key node up to and including the root node.

At first the set P starts out as empty and the graph only contains one node which is the root node. Nodes are added to the graph whenever a revoked receiver joins and nodes are removed from the graph whenever a privileged receiver revokes.

[10] For the origin of LKH see [And05 pg 12] or [OA05 pg 5]

If possible, the joining user u_i, with his individual key k_j, is just attached to one of the existing k-*nodes* at the second to lowest k-node depth in the key tree. However, if all k-nodes at that depth are full, i.e. already have d children, a new depth in the key tree must be created. See the Figure 3 for an example. As the new user is added as a leaf to the key tree, all keys on the path from the new user to the root are affected and must be updated.

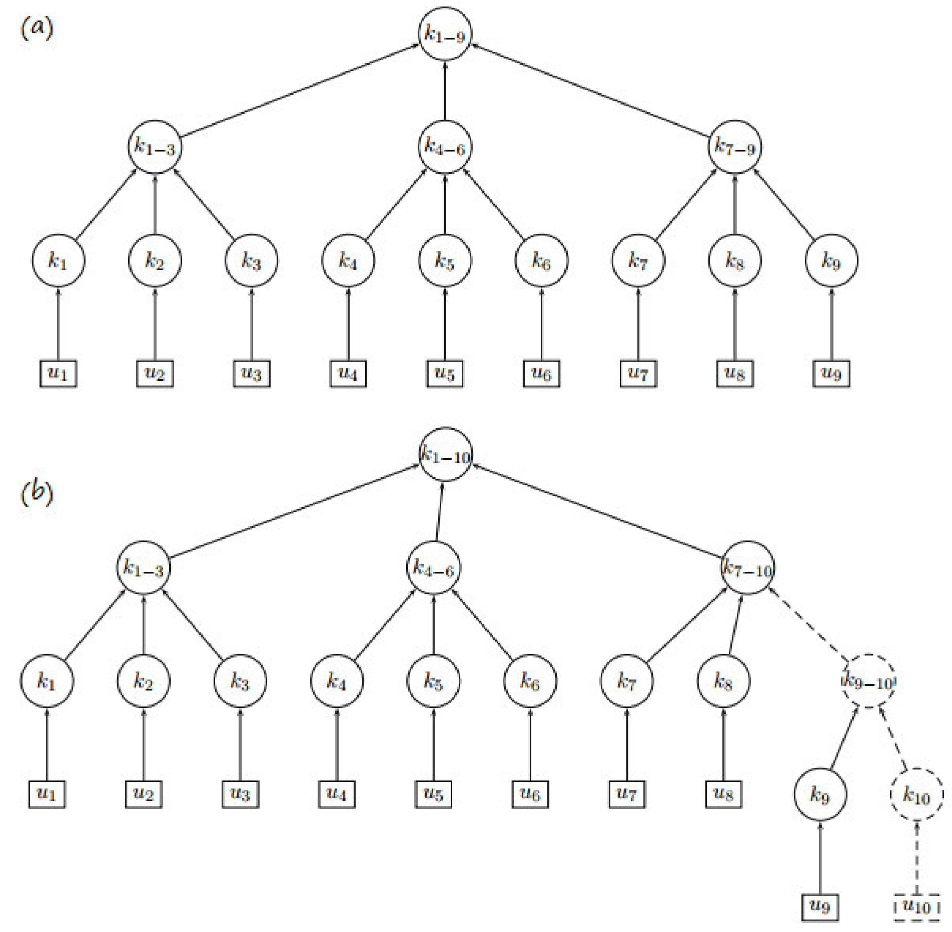

(a)
k_{1-9}
k_{1-3}
k_{4-6}
k_{7-9}
k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8 k_9
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9
(b)
k_{1-10}
k_{1-3}
k_{4-6}
k_{7-10}
k_{9-10}
k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8 k_9 k_{10}
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 u_{10}

Figure 3: How a user is added to LKH key tree.

In Figure 3, (a) is an example of a key tree with degree of 3, therefore the tree is complete (considering the k-nodes only). When a user u_{10} joins the privileged set, another k-node must be created as illustrated in (b) the dashed lines indicate new nodes and edges in the key tree.

When a user leaves the privileged set, his u-*node* and his individual k-*node* are removed from the key tree. In order to ensure that the leaving user cannot decrypt any future messages, all keys on the path between this

departing user's u-*node* and the root must be changed and updates sent to all remaining users. This enforces forward secrecy.

When a user joins the privileged set he must of course receive the appropriate keys in order to be able to decrypt future transmissions. However, LKH is also concerned with backward secrecy and therefore several of the previously used keys need to be updated as well.

Because keys are updated every time a user joins or leaves, LKH is resilient to any number of attackers from the non-privileged set.

Whenever a receiver $\bar{\underline{r}}$ joins $\bar{\underline{P}}$ or leaves $\bar{\underline{P}}$, the keys in the graph are updated to maintain both *forward* and *backward secrecy*.

2.2.1 Adding a revoked receiver to R

When a revoked receiver $r \in R$ wants to leave R and join P, then the following is done:

a) G authenticates r via some authentication protocol and ensures that r is allowed to join.
b) G removes r from R and adds it to P.
c) G generates a new leaf key node and assign it to r along with the keys from the path where r's key node is located and up to and including the root node.
d) To maintain backward secrecy and prevent r from decrypting previous broadcast, all the key nodes from r's key node location and up to and including the root node are regenerated and sent via a rekeying message to the current privileged receivers.

2.2.2 Removing a privileged receiver from P

When a privileged receiver $r \in R$ wants to leave P and join R, then the following is done:

a) G removes r from P and adds it to R.
b) G removes the leaf key node from the key tree.
c) To maintain forward secrecy and prevent r from decrypting future broadcast, all the key nodes from r's old key node location and up

to the root and including the root node are regenerated and sent via a rekeying message to the current privileged receivers.

2.2.3 Rekeying

Whenever a receiver joins or leaves the set of privileged receiver P, some keys are regenerated to maintain backward and forward secrecy. The process of B sending new keys to some privileged receiver is known as "rekeying". For more on the three different strategies on how to construct and send the rekey message, see (Obied, April 2005).

2.2.4 Encryption

Any broadcast secret $s \in S$ is encrypted with the key of the root node. If the key of the root node is K and the broadcaster B wants to broadcast a secret $s \in S$ then B broadcasts

$E(K, s) = C$

2.2.5 Decryption

Any broadcast secret $s \in S$ is decrypted with the key of the root node. If the key of the root node is K and the broadcast message was

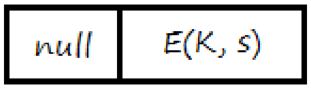
$C = E(K, s)$

then a privileged receiver can recover s by applying

$D(K, C) = D(K, E(K, s)) = s$

2.2.6 Broadcast message

It was mentioned before that any broadcast message has a header and a message body. In the LKH scheme, a broadcast message looks as follows:



The header in this scheme contains no information and the body contains the cipher text of the broadcast message.

2.2.7 Analysis and Complexity

One might wonder what is the point of keeping all the other keys if everything is encrypted with the root node key. Basically the other keys are used in the rekeying procedure to protect the key of the root node whenever it is regenerated and redistributed. Since all the privileged receivers know the key of the root node then it is quite efficient in terms of encryption and decryption. Table 2 shows the complexity of the LKH scheme in terms of big-O notation:

Table 2: the complexity of the LKH scheme

Storage Space	Message Size	Processing Time
$O(h)$	$O(1)$	$O(1)$

2.2.8 The trouble with LKH

The LKH has two drawbacks:

- The key tree can become rather unbalanced, after several joins and leaves have occurred, thus affecting the efficiency of the scheme. This will require balancing the key tree. For methods of balancing the key tree, see (Anderson, 2005).
- The other drawback in terms of performance issues of this scheme is the storage space of the keys. If the height of the key graph is h then a receiver must know all the keys from its assigned key node up to and including the root node.

2.3 Subset cover schemes

Subset cover schemes are a general class of stateless schemes. To learn where they were first introduced see (Johansson, Kreitz, & Lindholm, 2006). A subset cover algorithm predefines a family of subsets of users $F = \{f_n\}$, $f_i \subseteq U$. Each subset $f_i \in F$ is assigned a long-lived key k_i such that each $u \in f_i$ can compute k_i from its secret information I_u but any user $u \notin f_i$ cannot compute k_i. Because subset cover schemes are *stateless*, the sets F and the keys associated with its subsets are fixed.

Of these subsets, the ones that contain *members* are together known as *the cover* or *the subset cover* and the number of these subsets called the *cover size*. To distribute a new group key, the key server calculates an exact cover

$F_t = \{f_{t_m}\} \subseteq F$ and $\cup F_t = U / R = P$ i.e. a user is a member if and only

if the user is in P and $P \cap R = \phi$ (*null*).

The *Broadcast algorithm* at the Broadcast Center does the following:

- Choose a session encryption key K.
- Encrypt K separately m times with keys $k_{t_1}, k_{t_2}, \cdots, k_{t_m}$ and sends the ciphertext
  $\langle [f_{t_1}, f_{t_2}, \cdots, f_{t_m}, E(k_{t_1}, K), E(k_{t_2}, \langle\ ), \cdots, E(k_{t_m}, K)], F(K, S) \rangle$
  The portion in square brackets preceding is called the *header* and $F(K, S)$ is called the *body*. E and F are the encryption algorithms with their first parameter the encryption key

The *Decryption algorithm* at the receiver, upon receiving the broadcast message

$\langle [f_{t_1}, f_{t_2}, \cdots, f_{t_m}, C_1, C_2, \cdots, C_m], S' \rangle$
does the following:

- Find k_j such that $u \in f_{t_j}$ (in case $u \in R$, the result is null).
- Extract the corresponding key k_{t_j} from I_u.
- Compute $D\left(k_{t_j}, C_j\right)$ to obtain K.
- Compute $D(K, S')$ to obtain and output S.

The following section is a discussion on two related implementations of the subset cover scheme. In both schemes the subsets and the partitions are obtained by imagining the receivers as the leaves in a rooted *full and complete binary tree*[11] with N leaves (N is a power of 2). Such a tree contains $2N-1$ nodes (i.e. N leaves plus $N-1$ internal nodes) and for any $1 \le i \le 2N-1$ we assume that v_i is a node in the tree. The systems differ in the collections of subsets they consider. Like the LKH, the N receivers are each associated by a leaf in the key tree[12] - the key server maintains a perfect binary (key) tree and assigns a fixed position (a leaf in the key tree) to each distinct member.

[11] A *perfect binary tree*. Some people "wrongly" call this a *complete tree* while others call it a *full tree*. If it is not the case that $n = 2^k$ for some integer k then any *complete binary tree* with at least *n leaves* can be used. The extra *leaves* are then considered either as representing *privileged users* or as *non-privileged users*, whichever is the most favourable in the particular scenario. If *backward secrecy* is an issue and the extra leaves will be assigned to new *users* in the future then these *leaves* must be considered to represent *non-privileged users.*

[12] In this thesis, a leaf node in the key tree and a member assigned to that node are treated indistinguishable when there is no risk of ambiguity.

2.4 The Complete Subtree Method

The collections of subsets into which users belong $F = \{f_n\}$ corresponds to subtrees in the perfect binary tree. For any node v_i in the tree (either an internal node or a leaf, $2N - 1$ altogether), v_i is therefore an ancestor of each user in the subtree rooted at v_i. Let the subset T_i be the collection of receivers that correspond to the leaves of the subtree rooted at node v_i. In other words, $u \subset T_i$ iff v_i is an ancestor of u.

The key assignment method is simple: assign an independent and random key to every node in the complete tree and provide every receiver with the $log_2 N + 1$ keys associated with the nodes along the path from the root to leaf u; a user is given the key in the leaf node l_u it is associated with and all the keys in nodes that are ancestor of l_u. See the $Figure\ 3$. This ensures that all users whose corresponding leaves have a common ancestor also have a common key. In other words, the users whose leaves are in the subtree rooted at node v_i have a common key and they are said to belong to subset T_i. Thus all users belong to several subsets T_i.

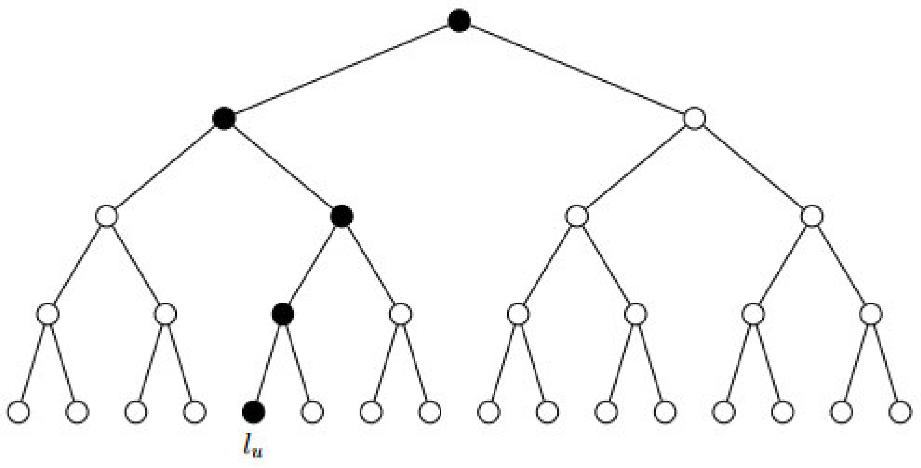


Figure 4: The keys given to a user

In the CS scheme (Figure 4), user u corresponds to a leaf l_u is given the keys from the root to the leaf l_u, the ones corresponding to the black leaves in the figure.

The method to partitions the $U \setminus R$ into disjoint subsets is as follows. For a given set of revoked receivers, let $u_1 u_2, \cdots, u_r$ be the leaves corresponding to the elements in R. A (directed) Steiner tree induced by a set $R, ST(R)$, is the minimal subtree of the perfect binary tree that connects all leaves in R and the root. See figure 5.

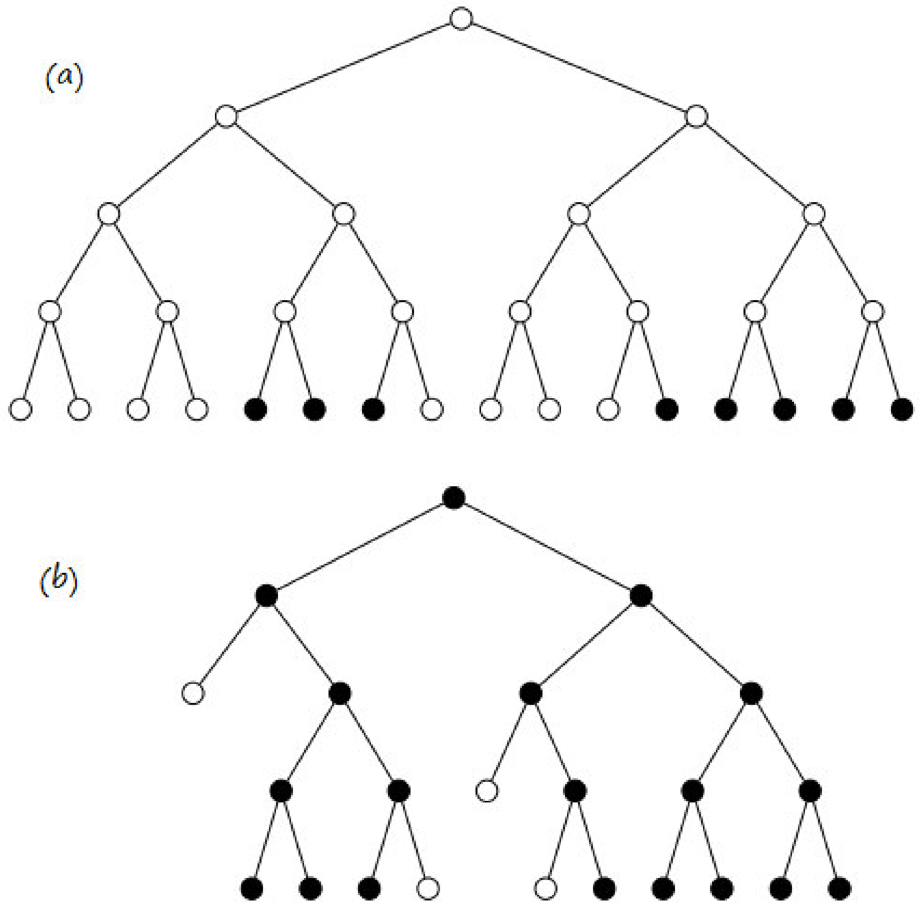


Figure 5: A Cover for the CS method generated from a Steiner tree.

In Figure 5, (a) is an example of a user configuration in the CS scheme, black leaves represent users in R and white leaves represent uses in P. The black nodes in (b) make up the Steiner tree generated from the user configuration in while white nodes are the ones hanging just off the Steiner tree. When using the CS scheme to send a secret to the user configuration in (a), it is thus the keys associated with the subtrees rooted at the white nodes in (b) that are used.

When this Steiner tree has been generated, the subsets F_i needed to cover the privileged set P can be found as follows; the subtrees hanging just off the Steiner tree are the subtrees $T_1, T_2, \cdots, T_m$ whose roots $v_1 v_2, \cdots, v_m$ are adjacent to nodes of outdegree 1 in $ST(R)$ but are not in $ST(R)$. These subtrees include all leaves corresponding to privileged users and only these users. For a proof of this, see (Naor, Naor, & Jeff, 2001). Therefore the keys to use are the ones associated with these subtrees.

The Decryption process proceeds as follows. Given a message

$\langle [T_1, T_2, \cdots, T_m, E(k_{T_1}, K), E(k_{T_2}, K), \cdots, E(k_{T_m}, K)], F(K, M) \rangle$

a receiver u needs to find whether any of its ancestors is among $T_1, T_2, \cdots, T_m$; note that there can be only one such ancestor, so u may belong to at most one subset. The user then extracts the corresponding k_T from their I_u and use it to extract K which they then use to extract M.

2.5 The Subset Difference Scheme

The collection of subsets $f_1, f_2, \cdots, f_m$ defined by this scheme corresponds to subsets of the form "a group of receivers G_1 minus another group G_2", where $G_2 \subset G_1$. The two groups G_1 and G_2 correspond to leaves in two perfect binary subtrees this way; a valid subset f is represented by two nodes in the tree v_i and v_j such that v_i is an ancestor of v_j and we denote such subset as $v_{i,j}$ and defined as $f_i \setminus f_i$ and referred to as *subset difference set*. A leaf $u \in f_{i,j}$ iff it is in the subtree rooted at v_i but not in the subtree rooted at v_j, or in other words $u \in f_{i,j}$ iff v_i is an ancestor of u but v_j is not i.e. $f_{i,j}$ is the set of all leaves in the subtree rooted at v_i, except for those in the subtree rooted at v_j. Thus in the SD scheme, a subset of users that have a common key is a *set difference* between two sets hence the term *set difference*. Figure 6 depicts the SD $f_{1,5}$.

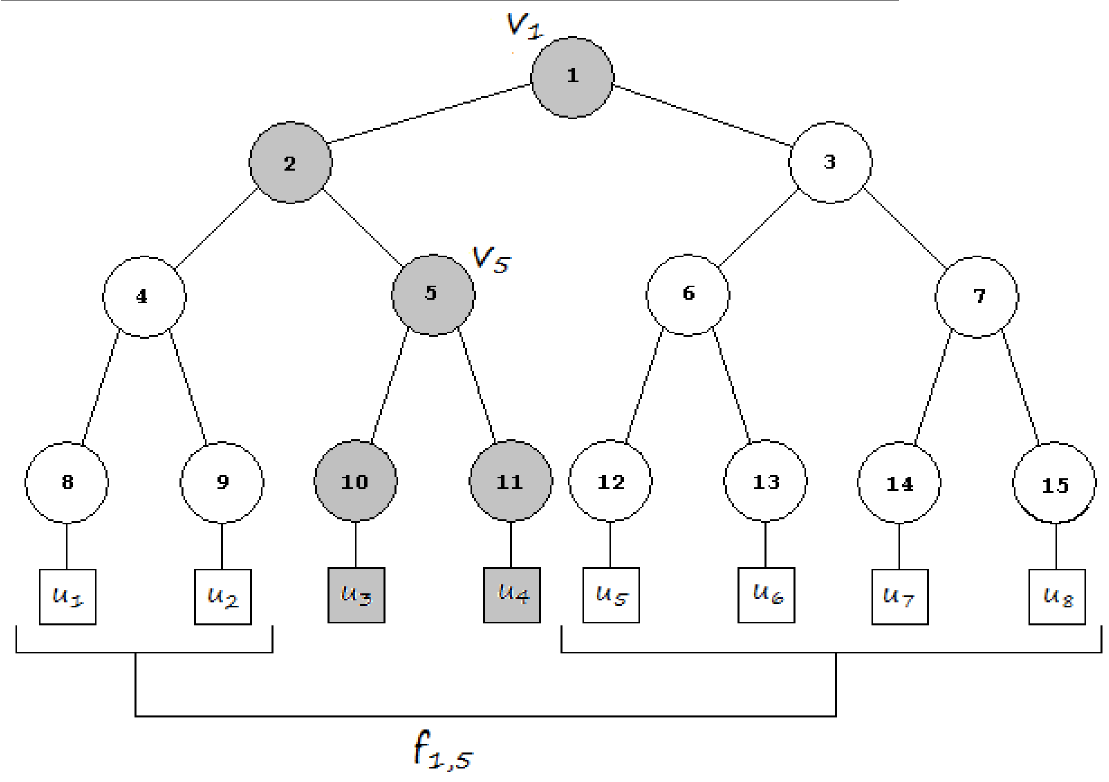


Figure 6: A Cover for the SDS generated from a Steiner tree.

An illustration of the SD $f_{1,5}$ is shown in Figure 6. Note that in CS, the *cover size* would have been 2 i.e. f_4. and f_3. Clearly, the subset difference scheme generates a smaller cover compared to complete subtree.

The goal of Subset Difference Scheme is to partition the $U \setminus R$ into fewer subsets than the CS scheme while still retaining the collusion resistance. However the way it achieves this, as we see in the next section, results in the number of keys stored by each receiver being greater.

Note that all subsets from the Complete Subtree Method are also subsets of the Subset Difference Method because they, too, can be expressed as a complete subtree minus another complete subtree; specifically, a subtree appears here as the difference between its parent and a sibling of the parent. The only exception is the full tree itself which has a special subset.

Each subset $f_{i,j}$ has an associated key $k_{i,j}$ and for how this is obtained, see next section.

2.5.1 The Cover

For a given set R of revoked receivers, let $u_1, u_2, \cdots, u_r$ be the leaves corresponding to the elements in R. The cover is a collection of disjoint subsets $F = \{f_{s_m} \setminus f_{t_m}\}$ which partitions $U \setminus R$. There are two algorithms presented in (Naor, Naor, & Jeff, 2001) for finding the cover. The one that the simulator (see section 2.7, The single Tree solution"), in this thesis implements is the second one of the two which uses maximal chains in the Steiner tree induced by the non-privileged leaves.

A *chain* in a Steiner tree is a set of nodes along a path where each node except the lowest one (the one with the largest depth) has exactly one child. A *maximal chain* is a chain that is not part of a longer chain. (Anderson, 2005) The start of a maximal chain is therefore the root or a node that has a sibling and the end is either a leaf (no child) or a node with two children. Also note that the shortest maximal chain consists of two nodes.

It finds a cover as follows:

- Generate the ST(R)
- For each *maximal chain* in the ST(R), add a subset $f_{i,j}$ where the top node in the chain is i and the bottom node is j.
  A maximal chain is obtained as follows:
  a maximal chain is a chain of nodes with outdegree 1 in ST (R) of the form $[v_{t_1}, v_{t_2}, \cdots, v_{t_m}]$
  - all of $v_{t_2}, v_{t_2}, \cdots, v_{t_{m-1}}$ have outdegree 1 in ST (R)
  - v_{t_m} is either a leaf or a node with outdegree 2
  - v_{t_1} is either the root or the child of a node of outdegree 2.

  For each such chain where $m > 1$, known as a nonempty chain, add a subset $v_{i_1} \setminus v_{i_m}$ to the cover. Note that all nodes of outdegree 1 in ST (R) are members of precisely one such chain.

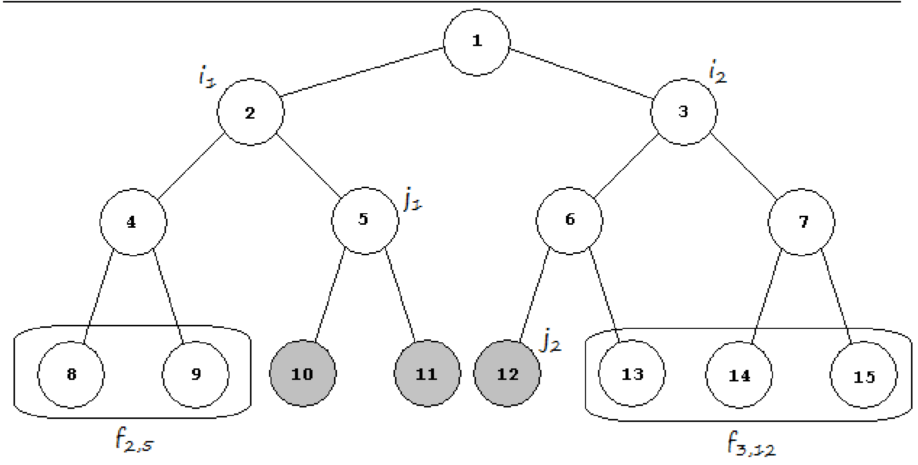


Figure 7: The SDS cover sets

Figure 7 illustrates a key tree with two SD sets of members. The active users are white leaves for (the privileged users) and black leaves are the revokes users.

It should be clear that a user in the SD scheme belongs to several SD subsets. In Figure 7 for example, the user associated with *k-node* 13 would belong to the SD subsets $f_{1,12}, f_{1,7}, f_{1,2}, f_{3,7}, f_{3,12}, f_{6,12}$. The figure 7 shows the Steiner tree corresponding to the privileged set in Figure 6. We can see in the diagram (Figure 7) that there is a maximal chain between nodes i_1 and j_1, and another between nodes i_2 and j_2, each corresponding to an SD subset.

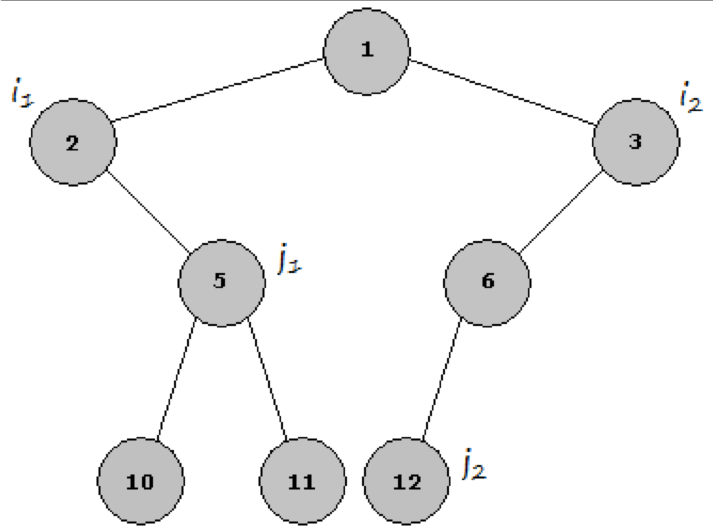


Figure 8: Maximal chains in a Steiner tree

The privileged users represented by the leftmost white leaves if Figure 8 can be reached by a transmission to $f_{i_1} \setminus f_{j_1}$ and these leaves are descendants of i_1 but not of j_1. The privileged users represented by the rightmost white leaves can similarly be reached by a transmission to $f_{i_2} \setminus f_{j_2}$.

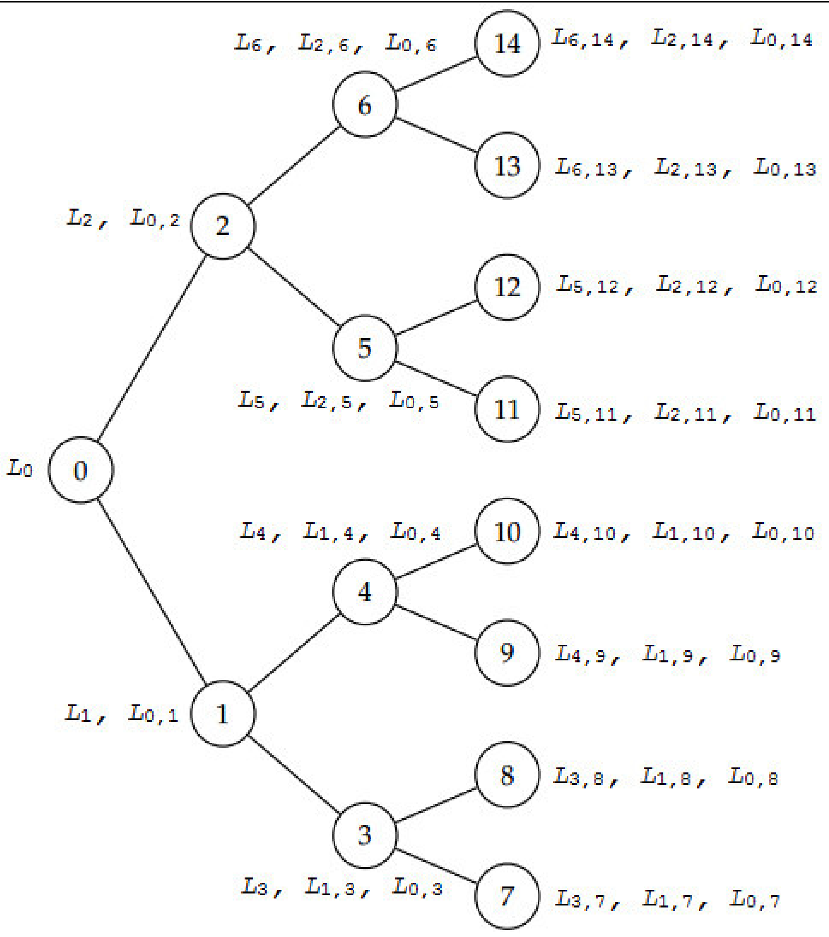
2.5.2 Key Assignment to subsets

The generation of these keys is done this way:
For each $1 \leq i \leq N$ corresponding to an *internal node* in the perfect binary tree, we choose a random and independent value L_i. i.e. the random label for node v_i is denoted L_i The initial labels L_i are used as the first input to a (cryptographic) pseudo-random sequence generator (one-way function) G[13] that triples the input, i.e. whose output length is three times the length of the input, that is
$G(L_i) = \langle G_L(L_i), G_M(L_i), G_R(L_i) \rangle$

[13] We say that G is a pseudo-random sequence generator $G:\{0,1\}^n \rightarrow \{0,1\}^{3n}$ if no *polynomial-time* adversary can distinguish the output of a randomly chosen seed from a truly random string of similar length.

Of these three outputs, two are used as labels for the children of node v_i, left third of the output, $G_L(L_i)$ becomes the label of the left child and the right third, $G_R(L_i)$ becomes the label of right child. The middle third of the output, $G_M(L_i)$, is used as a key corresponding to node v_i. From such a recursive top-down labelling process, given the label of a node, it is possible to compute the labels (and keys) of all its descendants i.e. each node v_j is associated with several labels $L_{i,j}$ each derived from the initial label of each of its ancestors v_i. Figure 9 shows in a small example which nodes have what labels associated with them.

Consider the subtree T_i (rooted at v_i) with the root assigned a label L_i. If v_j is a node in the subtree $T_i (i \neq j)$, then $L_{i,j}$ is label of node v_j derived in the subtree T_i i.e. from the random label of node v_i (by applying the function G one or more times), the key $K_{i,j}$ assigned to set SD set $f_{i,j}$ is $G_M(L_{i,j})$.



L_6, $L_{2,6}$, $L_{0,6}$
$L_{6,14}$, $L_{2,14}$, $L_{0,14}$
$L_{6,13}$, $L_{2,13}$, $L_{0,13}$
L_2, $L_{0,2}$
$L_{5,12}$, $L_{2,12}$, $L_{0,12}$
L_5, $L_{2,5}$, $L_{0,5}$
$L_{5,11}$, $L_{2,11}$, $L_{0,11}$
L_0
L_4, $L_{1,4}$, $L_{0,4}$
$L_{4,10}$, $L_{1,10}$, $L_{0,10}$
$L_{4,9}$, $L_{1,9}$, $L_{0,9}$
L_1, $L_{0,1}$
$L_{3,8}$, $L_{1,8}$, $L_{0,8}$
L_3, $L_{1,3}$, $L_{0,3}$
$L_{3,7}$, $L_{1,7}$, $L_{0,7}$
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

Figure 9: An illustration of how labels are generated in the SD scheme.

This top-down labelling process is continued so that for each child of node v_i the new label is used as input to G and labels and keys are created for all nodes in the subtree rooted at v_i. The result is that each node will be associated with several labels because the node is a part of several subtrees, except the root node which will only have one label.

The labels with one index, L_i, are the initial random labels assigned to each internal node v_i. The labels with two indices are derived from the initial labels such that label $L_{i,j}$ is the label at node v_j derived from label L_i. For example, $G_M = G_L(L_{1,4}) = G_L(G_R(L_1))$. The tree has been rotated in order to avoid too much clutter with all the labels.

What is the information I_u that each receiver gets in order to derive the key assignment described above? For each subtree T_i such that u is a leaf of T_i, the receiver u should be able to compute $L_{i,j}$ iff v_j is not an ancestor of u. Consider the path from v_i to u and let $v_{j_1}, v_{j_2}, \cdots, v_{j_k}$ be the nodes just "hanging off" the path, i.e. are adjacent to the path but not ancestors of u (one step sidewise from the path). Each node v_j in T_i that is not an ancestor of u is a descendant of one of these nodes. Therefore if u receives all the labels (derived, not the initial ones) of $v_{j_1}, v_{j_2}, \cdots, v_{j_k}$, as part of I_u, then invoking G (which is assumed to be known to u) at most $log\ N$ times suffices to compute $K_{i,j} = G_M(L_{i,j})$ for any v_j that is not an ancestor of u i.e. once a user has a label, the user can derive all the other labels down the tree.

How many labels in total are these per user? In each subtree that contains u, each of the nodes one step sidewise from the path between u and the root of the subtree contribute the d labels it acquires from its d ancestors where d is the depth of the node. Onto this we add one label for the case where there are no revocations for a total of:

$$1 + \sum_{d=1}^{d=h} k = \frac{(h+1)}{2} \times h + 1 = \frac{h^2}{2} + \frac{h}{2} + 1 = \frac{1}{2} log^2 n + \frac{1}{2} log\ n + 1$$

labels (Naor, Naor, & Jeff, 2001)..

Note: For a perfect tree the height, h of the tree and the number of nodes, n have the relationship:

$h = \log n$

2.6 A Multi Tree solution

The only work on dynamic subset difference Revocation in literature seem to be (Chen, Ge, Zhang, Kurose, & Towsley, 2004). The authors observe that:

- Static SDR generally requires a very large key tree, which must accommodate all unique members and as a consequence,
- currently active members, usually a small fraction of all of the unique members, are likely to be widely dispersed in the key tree space.

Both of these factors decrease the performance of SDR. It is these two inefficiencies that their proposed approach which they call dynamic SDR addresses.

They argue that, in the binary key tree of SDR, the probability of holes (positions with departed members) being somewhat distributed fairly evenly among members is high and therefore the cover size at most times will be made up of shorter subtrees rather than tall subtrees. The key server only needs to maintain a set of smaller subtrees with an appropriate height H. Their idea of dynamic SDR is to dynamically maintain such set of subtrees.

If each tree has a height of 0, each user key storage will be $O(1)$ because a user only need to store the key in it user key node because it has only one descendant. However, the message expansion will be maximum i.e. as many as the users because the broadcaster has to send the message to each user encrypted with the user's key stored in the user's key node.

The ideal case is when all the users are on one tree. Assuming that the users u, are a power of 2 and all of them are in privileged, if they all fit on one tree then the message expansion would be $O(1)$ one because they are all descendants of the root node and the key size at the users would be maximum because key size is $O(h)$ and the height is the maximum possible when the number of leaves are at a maximum.

To achieve this ideal case, requires that the number of users who will be active, be known in advance, something not possible. The authors have explained how to determining an appropriate height for each tree. It is a design trade off. (Chen, Ge, Zhang, Kurose, & Towsley, 2004).

2.6.1 Scheme of dynamic SDR

In dynamic SDR, the positions of members are not pre-assigned. Instead, the spaces in the key tree are dynamically allocated and reclaimed, adapting to the current set of active members. More specifically, the key server dynamically creates leaves when new member joins, or discards a subtree when all positions of the subtree are inactive. By doing this, the key server maintains active members in a dynamic key tree, rather than a large key tree constructed in advance.

The scheme is fairly simple. The small equally sized disjointed trees are atomic allocation units. One can view them as subtrees connected to a virtual root r (see Figure 10).

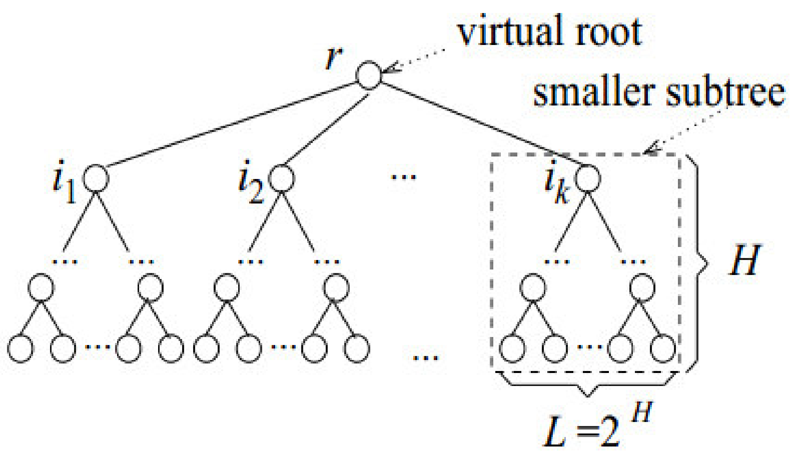


Figure 10: Constructing dynamic SDR key tree using sub-trees as allocation units

Initially, the key server has a single subtree T_i connected to the virtual root r.

When a member joins the group, regardless of being a new member or a returned member, the member is assigned to the next available position in the key tree (from left to right) and is unicast the secret information associated with the new position. The key server thereafter encrypts and multicasts the updated TEK to the current members in exactly the same way as in static SDR. If new positions are required the key server creates a new subtree T_k. When a member, m, leaves the group, the position becomes empty and will never be used by any member (even m itself) and a new TEK is multicast to the members that remain in the group. If all positions of the leftmost subtree become empty, the key server discards that subtree.

2.6.2 The benefits

The advantages of maintaining such a dynamic-membership key tree are two-fold.

First, instead of maintaining a key tree that is sufficiently large to hold all potential members, dynamic SDR may require a much smaller key tree of a size sufficient to accommodate the maximum number of concurrently active members. This helps reduce key storage cost, both at the members and at the key server. Second, dynamic SDR is able to utilize the temporal locality of the members' joining and leaving activity. By assigning members that arrive close in time to positions that are close in the key tree, the key server is likely to find a subset that can cover many adjacent members. As a result, a small number of subsets will typically be needed to cover the active members when a new TEK is disseminated. This implies that the messaging overhead associated with rekeying is also reduced.

Since the key tree of dynamic SDR can be extended arbitrarily, dynamic SDR does not require a priori knowledge of the size of total member population, N. This avoids the problem, which exists in static SDR, of estimating N. Overestimating N makes the static SDR key tree unnecessarily large, increasing both rekey communication cost and key storage cost, whereas, underestimating N may introduce the problem of having to reject members when all positions have been assigned.

2.6.3 Key Storage

In dynamic SDR, the size of secret information, $|I_u|$, is reduced from $(log^2 N + log\ N)\ /\ 2 + 1$ to $(log^2 L + log\ L)\ /\ 2 + 1$ where L is the maximum number of users per subtree that corresponds to an allocation unit. Although the key storage size required by a member is fixed when L is chosen, this is not the case for the broadcaster B, whose key storage is related to the number of subtrees. $L = 2^h$ i.e. the number of leaves in a tree whose height is h is 2^h.

2.6.4 Reducing S by shifting

Assuming that the key server sequentially assigns the available positions of the key tree from the left to the right to the joining members, S is the distance from the leftmost position occupied by an active member to the first available position at the right side. These S positions, referred to as the concurrent spaces, determine the key storage at the key server.

A large value S results in an increased key storage at the B side. Also, currently active members disperse in the key tree as S increases, incurring more resultant subsets and thus more rekeying messages. As a result, it is desirable to keep S small.

A simple operation, namely *shifting*, that can be used to reduce S. This defined as the operation of detaching the leftmost active member in the key tree and reattaching the member to the next available position (for new arrivals) in the key tree. This is based on the consideration that in a dynamic SDR subtree, a leaf position assigned to a member cannot be assigned again. See Figure 11.

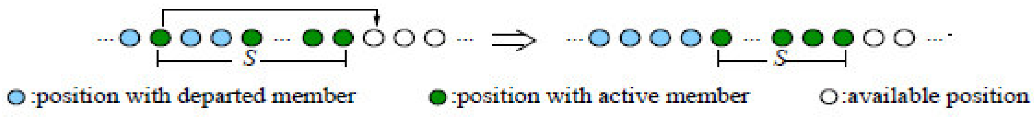


Figure 11: The Shifting operation

When some holes (i.e. positions with departed members) are generated in the key tree, shifting the leftmost active member may reduce the concurrent spaces, $\overline{\underline{S}}$, and make active members more adjacent. Here we see shifting the leftmost member from left to right reduces $\overline{\underline{S}}$ by $\overline{\underline{2}}$.

When active members are shifted, they are delivered new secret information associated with the new positions by unicast. From the collusion-proof property of static SDR, shifting does not jeopardize the confidentiality of the group communication.

2.7 The Single Tree Solution

The solution proposed in this thesis uses a single tree that grows and shrinks as users come and go respectively. The solution proposed in in (Chen, Ge, Zhang, Kurose, & Towsley, 2004) uses a set of sub-trees instead. Like in (Chen, Ge, Zhang, Kurose, & Towsley, 2004), when a new member joins the group, the member is assigned to the next available position in the key tree (from left to right) and is unicast the secret information, I_u associated with the new position. But unlike unlike (Chen, Ge, Zhang, Kurose, & Towsley, 2004) where a returned member, is also assigned to the next available position in the key tree, the single tree returns the user to their former position - a member reoccupying their former position does not compromise the security of the scheme so long as rekeying is done. The rekeying is necessary because the return of a member to the same position they were in changes the cover.

All the time, the users are accommodated on a single key tree whose size is the smallest possible that can sufficiently accommodate them. This achieved as follows:

- When a user leaves, the systems checks if the remaining members are equal to or less than half the base of the tree. If so, this means they can be accommodated on a smaller tree and the system destroys the current tree and creates a new smaller tree sufficient to accommodate them.
- When a new user or a previously revoked user comes, and there is no slot to place them, the system creates a new tree that is sufficient to accommodate the current members. The resulting tree will not

necessarily be bigger because lack of a new slot does not necessarily mean the tree is full; some users may have left and a user slot can only be occupied by the user who is initially allocated the slot. Note that previously revoked user may lack a slot in the tree because the tree from which they were revoked has since been destroyed and this is an entirely different tree.

Keeping users on the smallest possible tree ensures that the key storage cost at both B and at the members is always the lowest possible. The downside of the solution is that a unicast is made to each member whenever the tree is recreated; a high communication cost. Indeed, the performance of this solution depends on how infrequently these two situations occur.

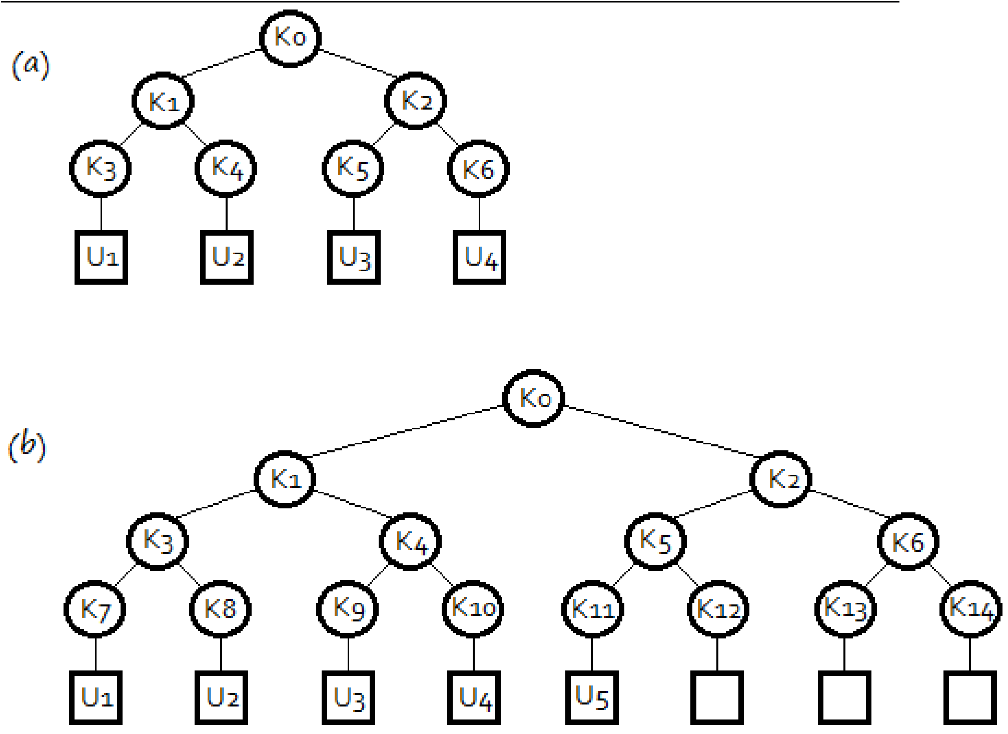


Figure 12: Adding a user to a keytree that is already full of members

In Figure 12, the tree in (a) is full of members (4 members) while the tree in (b) is the resulting tree when a new member u_5 joins. Note that if one member leaves the tree in (b), the system creates a new tree of the same size as the one in (a). Also note that a key K_i in a newly created key tree is not related to the key K_i in the previous tree. This is why the I_u must be unicast to each user each time the key tree is created.

To keep the broadcast message expansion low, the tree should be re-created whenever the number of *user clusters* is "too high". Consider the n leaves of the key tree as a sequence of users $U = [u_1, u_2, \cdots, u_n]$ that can be thought of as a bitmap where a 0 at position i means that the corresponding user, u_i, is in the non-privileged set and 1 means that u_i is in the privileged set. The sequence U is called a user profile. A cluster is an unbroken sequence of members (1s) in U. The number of clusters of members determine the cover size – the number of transmissions or the cover size.

2.7.1 Clusters and transitions

The maximum number of clusters C_{max} is reached when the user profile, U contains the most number of 01 or 10 subsequences which is clearly half the base of the tree (the maximum possible number of users on the tree). This is denoted as T_{max} where T stands for transition from the interpretation of 01 or 10 as a transition in the user profile from a non-privileged user to a privileged user or from a privileged user to a non-privileged user respectively. C_{max} and T_{max} are the same – they are both equal in value to half the number of maximum users possible. When T_{max} has been reached, then we have C_{max} but we can have C_{max} when the transitions are not yet T_{max}. T_{max} and C_{max} coincide only when each member (a 1 in U) has exactly one non-member neighbour (a 0 in U) and vice versa if the first and last bits in U are opposite bits; if both the last and the first members of U are 1s, then C_{max} is reachable when one and only one of the $0s$ has a 0 as a neighbour or one and only one of the $1s$ has a 1 as a neighbour and therefore the transitions are less than T_{max}. Note that C_{max} is not reachable if both the last and the first are $0s$ (non-members).

Because the system intervenes whenever the current members become half the maximum possible, we can never reach T_{max} but the explanation in the last paragraph shows it is possible to reach C_{max} before the system intervenes. Chances that C_{max} is reachable are very slim indeed – only when the last and the first user in the user profile are members and every *1* has a *0* as a neighbour in the profile except one and only one member is having a member as a neighbour therefore resulting in members being one more than half the maximum possible. Any other way of reaching C_{max} requires that the members reduce to half the maximum possible and as we have seen, whenever this would be the situation, the system intervenes and creates a smaller tree with one cluster of users.

Nevertheless, operating at C_{max} should be considered unacceptable and therefore, when this would be the situation, the system should intervene and re-create a smaller tree with one cluster of users. The relationship between the cover size and the number of user clusters is not one-to-one. If left to the system to intervene only when the members is half the maximum possible, C_{max} will be reachable as the Table 3 shows; it is a table of the values of user clusters and cover size for a key tree whose base is 16 users. The base of the tree is assumed here to be a zero-based array of size 16 (The indices are from 0 to 15). The last column of Table 3 and Table 4 shows the situation when the system has intervened. Table 3 is one in which revocations produce the 01 subsequences while Table 4 is the one in which revocations produce the 10 subsequences.

Table 3: the "01" subsequences

revoked(index)	0	2	4	6	8	10	12	14
cover size	1	3	4	5	5	7	7	1
member clusters	1	2	3	4	5	6	7	1
members	15	14	13	12	11	10	9	8

Table 4: the "10" subsequences

revoked(index)	1	3	5	7	9	11	13	15
cover size	1	3	4	5	5	7	7	1
member clusters	2	3	4	5	6	7	8	1
members	15	14	13	12	11	10	9	8

From the tables, one can see a possibility of reaching C_{max} before the system intervenes (see table two); the user clusters has reached the maximum of 8 while the users have not reduced to 8. As pointed out already, high user clusters means the system is performing badly in terms of message expansion.

As already pointed out, the performance of the single tree solution depends on how infrequently the tree creation operation takes place because among other things, it involves unicasting the I_u to each member which is a high cost in communication terms. The tree creation operation takes place when the following situations show up:

- admitting a member is going to result in all the users (*revoked* and *members*) being more than a power of two; the base of the key tree is full.
- revoking a *member* (privileged user) is going to result in *members* being half the maximum possible; half the key tree base.
- revoking or admitting a *member* is going to result in user clusters being the maximum possible (half the tree base)

With these constraints, we have a guarantee that the maximum message expansion (broadcast transmissions per message) can never reach the maximum possible i.e. half the base of the key tree. The downside is that whenever these situations show up, the key server is 'frozen' i.e. not available to be queried for broadcast key(s). It is therefore how frequent these situations can occur during operation that determines the efficiency of the scheme.

During certain intervals perhaps when there are no essential activities and the performance is not so good due to user fragmentation; the tree could be optimised by moving the members to a new tree in the process consolidating them into one cluster to reduce the cover size. Indeed whenever the key tree is created all the users are arranged in one cluster to the left as possible. This keeping of broadcast transmissions per message below a certain level and ensuring that the key storage at the broadcaster, B and at the users are at their minimal level are the key design strategies of the single tree solution of the Dynamic Revocation Problem.

2.7.2 <u>Summary</u>

The design goals of the scheme are ensuring that message expansion is always less than the maximum possible, and the key storage at the broadcaster, B and the users are always at their minimal possible level. As pointed out, this comes at a cost of the inevitable maintenance communication cost – unicasts of I_u to members. Whenever these maintenance activities are taking place, the key server is 'frozen' i.e. not available to be queried for broadcast key(s) by the broadcaster.

The maintenance activities are mainly reconstruction of the key tree which then requires among other things, a unicast of I_u to each member. The tree creation operation takes place when the following situations show up:

- admitting a member is going to result in all the users (revoked and members) being more than a power of two; the base of the key tree is full.
- revoking a member (privileged user) or deleting a member is going to result in members being half the maximum possible; half the key tree base.
- revoking/deleting or admitting a member is going to result in user clusters being the maximum possible (half the tree base)

The conceptual diagram is shown in Figure 13.

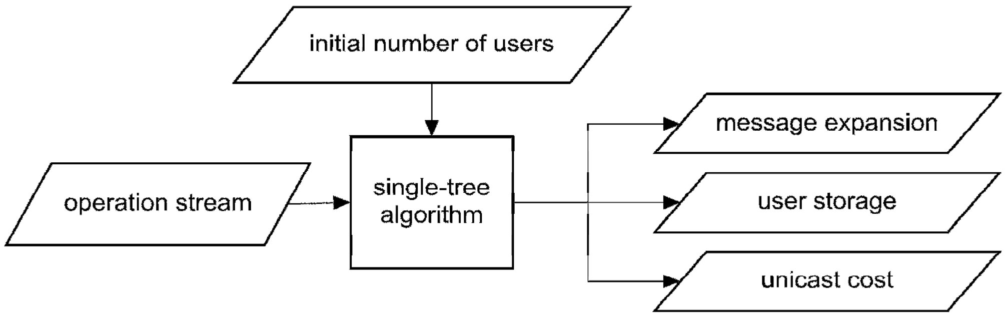


Figure 13: The Conceptual Diagram

Chapter 3: Methodology

Methodology is the path to finding answers to the research questions constitutes. The methodology used in this thesis is *simulation*. It is described in detail in Section 3.4 but before that, some background information is discussed that illustrate that simulation is a combination of actually two methodologies – experiment and model.

3.1 Research Philosophy

In an academic context, research is used to refer to the activity of a diligent and systematic inquiry or investigation in an area, with the objective of discovering or revising facts, theories, applications etc. Research is undertaken within most professions. More than a set of skills, it is a way of thinking: examining critically the various aspects of one's professional work. It is a habit of questioning what one does, and a systematic examination of the observed information to find answers with a view to instituting appropriate changes for a more effective professional service. The goal is to discover and disseminate new knowledge.

Research is a process of collecting, analyzing and interpreting information to answer questions. But to qualify as research, the process must have certain characteristics: it must, as far as possible, be controlled, rigorous, systematic, valid and verifiable, empirical and critical.

3.2 The Research Process

The research process is similar to undertaking a journey. For a research journey there are two important decisions to make

- What one wants to find out about or what research questions (problems) one wants to find answers to and
- How to go about finding these answers.

There are practical steps through which a researcher must pass in their research journey in order to find answers to their research questions. The path to finding answers to the research questions constitutes *research methodology*. At each operational step in the research process the researcher is required to choose from a multiplicity of methods, procedures and models of research methodology which will help the researcher to best achieve their objectives.

There are several methods that can be used in *Computer Science* and *Information Systems*. Tasks performed by a single researcher fall within different methodologies. Even the activities required to tackle a single research question may include several of these methodologies. [Ama]

The remainder of this section is a discussion of these methodologies.

3.2.1 Theoretical/Formal Methodology

The theoretical approaches to *Computer Science* are based on the classical methodology since they are related to logic and mathematics (Ayash, 2014). In Computing Science, formal methodologies are mostly used to prove facts about algorithms and system. Researchers may be interested on the formal specification of a software component in order to allow the automatic verification of an implementation of that component. Alternatively, researchers may be interested on the time or space complexity of an algorithm, or on the correctness or the quality of the solutions generated by the algorithm.

A formal methodology is most frequently used in theoretical Computing Science. *Theoretical Computer Science* (TCS) is formal and mathematical and it is mostly concerned with modelling and abstraction. The idea is to abstract away less important details and obtain a model that captures the essence of the problem under study. This approach allows for general results that are adaptable as underlying technologies and application changes, and that also provides unification and linkage between seemingly disparate areas and disciplines. TCS concerns itself with possibilities and fundamental limitations. Researchers in TCS develop mathematical techniques to address questions such as the following. Given a problem, how hard is it to solve? Given a computational model, what are its limitations? Given a formalism, what can it express? (Amaral, 2014)

3.2.2 Build Methodology

A "build" research methodology consists of building an artifact — either a physical artifact or a software system — to demonstrate that it is possible. To be considered research, the construction of the artifact must be new or it must include new features that have not been demonstrated before in other artifacts. (Amaral, 2014)

3.2.3 Process Methodology

A process methodology is used to understand the processes used to accomplish tasks in Computing Science. This methodology is mostly used in the areas of Software Engineering and Man-Machine Interface which deal with the way humans build and use computer systems. The study of processes may also be used to understand cognition in the field of Artificial Intelligence. (Amaral, 2014).

Process methodologies are most useful in the study of activities that involve humans. Examples of such activities in Computing Science include the design and construction of software systems — large or small, the design and evaluation of human-computer interactions, and the understanding of cognitive processes. More recently the creation of interactive games has been studied extensively. This activities often involve studies with human subjects.

3.2.4 Experimental Methodology

Experiments can test the veracity of theories. This method within CS is used in several different fields like artificial neural networks, automating theorem proving, natural languages, analysing performances and behaviours, etc. Experimental methodologies are broadly used in CS to evaluate new solutions for problems. Experimental evaluation is often divided into two phases. In an exploratory phase the researcher is taking measurements that will help identify what are the questions that should be asked about the system under evaluation. Then an evaluation phase will attempt to answer these questions. A well-designed experiment will start with a list of the questions that the experiment is expected to answer.

It is important to emphasize that all the experiments and results should be reproducible. Conducting experiments in a careless fashion can lead to a situation where the authors themselves cannot reproduce the experiments. The following is some general advise to help preventing one from producing worthless experimental papers. (Ayash, 2014).

3.2.5 Model Methodology

The real world can be viewed as being composed of systems. A *system* is a set of related components or entities that interact with each other based on the rules or operating policies of the system.

Oftentimes, there arises a need to predict some aspect of the performance of a system before it is actually built. Since the real machine does not yet exist, one cannot measure its performance directly (Lilja, 2000). Instead, the best one can do is to model the system. A model is a representation of the system. Modeling may be necessary because a system does not physically exist or building a system is expensive or measuring (analysing) a system is time-consuming requiring vast computing resources.

Models enable seeing how a real-world activity will perform under different conditions and test various hypotheses at a fraction of the cost of performing the actual activity. Modeling may also be appropriate when one wants to investigate some aspect of a system's performance that one cannot easily measure directly or indirectly.

Modeling is the purposeful abstraction of a real or a planned system with the objective of reducing it to a limited, but representative, set of components and interactions that allow the qualitative and quantitative description of its properties. Scientists build models that capture important aspects of a system and gloss over — either completely ignore or just approximate — the aspects that have lesser (or no) impact to their intended study. A *model* is an abstracted and simplified representation of a system at one point in time. Models are an abstraction because they attempt to capture the realism of the system. They are a simplification because, for efficiency, reliability, and ease of analysis, a model should capture only the most important aspects of the real system.

The decision of which aspects are important and which ones have lesser impact is itself part of the modeling strategy. Misleading outcomes are produced by models that eliminate what is important or that over-emphasize what is of lesser impact. (Amaral, 2014). The finer the level of granularity at which the model can simulate the system depends on the level of details necessary in order to make the desired decision and the consequences of being wrong; deciding the level of detail necessary is more art than science (Lilja, 2000). If the model is valid, the outputs of the simulation will be reflective of the performance or behaviour of the real system.

Because a model is much less complex than the system that it models, it allows the researcher to better understand the system and to use the model to perform experiments that could not be performed in the system itself because of cost or accessibility. Strictly speaking, modeling is a methodological aspect of science. Modeling is not the object of the research, it is part of an arsenal of instruments used by researchers to study and understand the research's object. (Amaral, 2014). The model is studied as a surrogate for the actual system.

One of the principal benefits of a model is that one can begin with a simple approximation of a process and gradually refine the model as their understanding of the process improves. This "step- wise refinement" enables one to achieve good approximations of very complex problems surprisingly quickly. As one adds refinements, the model more closely imitates the real-life process.

The model methodology is often used in combination with the other four methodologies. Experiments based on a model are called *simulations* (Amaral, 2014). From the simulations information, one can extrapolate how the system will behave once it is actually built (Lilja, 2000).

Model methodology is used especially in Computer Science not only because it offers the possibility to investigate the systems that is under invention or construction but also systems or regimes that are outside of the experimental domain. Normally complex phenomena that cannot be implemented in laboratories for example evolution of the universe. Some domains that adopt computer simulation methodologies are sciences such as astronomy, physics or economics; other areas more specialized such as the study of non-linear systems, virtual reality or artificial life also exploit these methodologies. A lot of projects can use the simulation methods, like the study of a new developed network protocol. To test this protocol one has to build a huge network with a lot of expensive network tools, but this network can't be easily achieved. For this reason we can use the simulation method.

Simulation often lacks the power to make definite statements about properties of the system. For instance, the results of simulations may not be used to prove that a deadlock never develops in a concurrent system.

3.3 Simulation Methodologies

The formalism used to specify a system is termed a modeling methodology. The three main modeling methodologies are *continuous*, *discrete event*, and *discrete rate*.

- Continuous modeling (sometimes known as process modeling) is used to describe a flow of values.
- Discrete event models track unique entities.
- Discrete rate models share some aspects of both continuous and discrete event modeling.

In all three types of simulations, what is of concern is the granularity of what is being modeled and what causes the state of the model to change.

3.3.1 Continuous

The time step is fixed at the beginning of the simulation, time advances in equal increments, and values change based directly on changes in time. In this type of model, values reflect the state of the modeled system at any particular time, and simulated time advances evenly from one time step to the next. For example, an airplane flying on autopilot represents a continuous system since its state (such as position or velocity) changes continuously with respect to time. Continuous simulations are analogous to a constant stream of fluid passing through a pipe. The volume may increase or decrease at each time step, but the flow is continuous.

3.3.2 <u>Discrete Event</u>

The system changes state as events occur and only when those events occur; the mere passing of time has no direct effect on the model. Unlike a continuous model, simulated time advances from one event to the next and it is unlikely that the time between events will be equal. A factory that assembles parts is a good example of a discrete event system. The individual entities (parts) are assembled based on events (receipt or anticipation of orders). Using the pipe analogy for discrete event simulations, the pipe could be empty or have any number of separate buckets of water traveling through it. Rather than a continuous flow, buckets of water would come out of the pipe at random intervals.

3.3.3 <u>Discrete rate</u>

Discrete rate simulations are a hybrid type, combining aspects of continuous and discrete event modeling. Like continuous models they simulate the flow of "fluid" rather than items; like discrete event models they recalculate rates and values whenever events occur. Using the pipe analogy for a discrete rate simulation, there is a constant stream of fluid passing through the pipe. But the rates of flow and the routing can change when an event occurs.

3.4 The Model: A Discrete-Event Model

The simulator implemented for this thesis is a *Discrete-Event Model. Discrete-Event Simulation* relies on a transaction-flow approach to modeling systems. A system is a collection of entities (Law, January 2014) that act and interact together toward the accomplishment of some logical end. A DES is used to model a system whose global state changes as a function of time. The basic idea is that the global state is appropriately updated every time some event occurs.

- Discrete System

  In a discrete system, state variables change instantaneously at separated point in time, e.g., a bank, since state variables (number of customers), change only when a customer arrives or when a customer finishes being served and departs.
- Continuous System

  In a continuous system, state variable change continuously with respect to time, e.g., airplane moving through the air, since state variables - position and velocity change continuously with respect to time

As already explained above and in Section 3.4, *discrete-event simulation* (DES) models the operation of a system as a discrete sequence of events in time i.e. a system whose state may change only at discrete point in time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation can directly jump in time from one event to the next. The events may be generated may be generated externally to the model as well as spawned within the simulator by the processing of other events.

Discrete simulators are generally designed for simulating processes such as call centers, factory operations, and shipping facilities in which the material or information that is being simulated can be described as moving in discrete steps or packets. They are not meant to model the movement of continuous material (e.g., water) or represent continuous systems that are represented by differential equations.

In the simulator implemented for this thesis, an event is the successful execution of an operation from the operation queue. While the specific details of every model will be unique, DESs all share a similar overall structure. In addition to the logic of what happens when system events occur, each DES will require at least some of the following components. (Lilja, 2000). This section describes the simulator, showing that it has all the characteristics of a generic Discrete-Event Simulator.

3.4.1 State

A system state is a set of variables that captures the salient properties of the system to be studied. The state trajectory overtime S(t) can mathematically represented by a step function whose values change in correspondence of discrete events. These are the dependent variables as explained in the, Section 3.6, Design of Experiment. The ones of interest in this thesis are three

- Unicast cost
- Multicast cost and
- User storage

But the simulator actually implemented for this thesis also records the extraneous value of *event duration* in clock ticks. See a sample of the detailed output in Appendix 2.

3.4.2 <u>An Events Scheduler</u>

The event scheduler maintains a list of all pending simulation events in their global time order. This list is sometimes called the pending event set because it lists events that are pending as a result of previously simulated event but have yet to be simulated themselves. It is the responsibility of the scheduler to process the next event on the list by removing it from the list and dispatching the event to the appropriate event-processing routine. It also inserts new events into the appropriate point in the list on the basis of the time of the time at which the event is supposed to be executed. Updates to the global time variable also are coordinated by the scheduler.

In this case, the list is a list of operations from a text file. As part of the construction of the class, **StringReader**, whose object tracks this pending list, the file's content is read.

3.4.3 <u>Clock: The global time variable</u>

The global time variable records the current simulation time. The simulation must keep track of the current simulation time, in whatever measurement units are suitable for the system being modeled. In discrete-event simulations, as opposed to real-time simulations, time 'hops' because events are instantaneous – the clock skips to the next event start time as the simulation proceeds. It can be updated by the scheduler.

In this case, this is the serial number of the operation with the first number given the serial 1. The way the experiment is designed is such that it reads 12 operations. See Section 3.6, Design of Experiment. So the reading routine apart from recording the serial number (clock) also ensures that it does not overstep.

3.4.4 <u>Event processing</u>

Each kind of event in the system will typically have its own event-processing routine to simulate what happens when that event occurs in a real system. These routines may update the global state and they may generate additional events that must be inserted into the pending-event list by the scheduler. The processing of each event depends entirely on the system being simulated though.

In this case, each solution has its own way of processing an event. For the single-tree solution, every event processing involves testing if the key-tree has tipped over the cliff. If so, a new tree is created as part of the event processing. See Section 3.6, Design of Experiment.

3.4.5 Event generation

Discrete event simulators are often classified according to the technique used to generate events. Commonly used classifications are execution driven, trace driven, and distribution driven.

The simulator for this thesis obtains the events from a prepared text file. There are twelve of them as explained in Section 3.6, Design of Experiment.

3.4.6 Statistics: Recording and summarisation of data

In addition to maintaining the state variables necessary for simulating the system, the model must also maintain appropriate event counts and time measurements. These values are used at the end of the simulation to calculate appropriate statistics to summarize the simulation results.

In this case, the statistics are simply the average of each variable values as recorded after each clock step (event). The summing is also done after each event. These averages appear at the end of the output. See a sample in Appendix 2.

3.4.7 Ending condition

Because events are bootstrapped[14], theoretically a discrete-event simulation could run forever. So the simulation designer must decide when the simulation will end. Typical choices are "at time t" or "after processing n number of events" or, more generally, "when statistical measure X reaches the value x"

In this case the operations to read have been set at 12 therefore after "time" 12 it stops.

3.4.8 Simulation Algorithm or Engine Logic

The overall processing done by a DES can be summarised in the steps:

```
Start

Initialize the following
   - number of operations to process (Ending Condition).

   - the dependent variables(the system state variables).

   - operation serial number (Clock - to zero).
```

[14] Bootstrapping usually refers to the starting of a self-sustaining process that is supposed to proceed without external input. In computer technology the term (usually shortened to booting) usually refers to the process of loading the basic software into the memory of a computer after power-on or general reset, especially the operating system which will then take care of loading other software as needed.

```
     - the cumulative statistical variables (to zero)
  while operation serial < number of operations to process
     Pick operation from list (remove next event from pending list)
     dispatch the operation to appropriate routine
     increase each cumulative statistical variables
          by corresponding updated state variables value.
     increment operation serial by one (i.e. update global time)
  Compute the averages (Generate statistical report).
```

3.5 Validation and Verification

The quality of the results obtained from any simulation of a system is fundamentally limited by the quality of the assumptions made in developing the simulation model (Lilja, 2000). Verification and validation are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it fulfils its intended purpose. The words "verification" and "validation" are sometimes preceded with "Independent" (or IV&V), indicating that the verification and validation is to be performed by a disinterested third party.

3.5.1 Validation of the simulator

Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s). The validation process attempts to ensure that the simulator accurately models the desired system. In other words, validation attempts to determine how close the results of the simulation are to what would be produced by an actual system (Lilja, 2000).

The simulator when run has an execution path for validating it known as the "Work Bench". When the user gets into this branch of execution, the outputs are displayed on the screen for inspection. The user can perform all the operations and watch their output; the operation menu is at the bottom and the output of pervious operation at the top. It is interactive; the user chooses an operation and sees the outcome. See Section 3.7.

3.5.2 Verification of the simulator

Verification is the process of determining that a computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer's conceptual description and specifications. Verification is the process of determining that a model is implemented correctly (Lilja, 2000).

The adherence of the simulator to the structure of a discrete event simulator is explained in Section 3.4.

3.6 Design of Experiment

In general usage, design of experiments (DOE) is the design of any information-gathering exercises where variation is present, whether under the full control of the experimenter or not. The primary goal of design of experiment is to determine the maximum amount of information about a system with the minimum cost. The cost includes the time and effort required to gather the necessary data, plus the time and effort needed to analyse these data to draw some appropriate conclusions. Consequently, it is important to minimise the number of experiments that must be performed while maximising the information obtained. (Lilja, 2000).

From the resulting measurements obtained, from the carefully selected experiments, the experimenter can determine the effects on performance of each individual input *factor* and the effects of their interactions.

The simplest design for an experiment varies one input (factor) while holding all other inputs constant. The experiments carried out for this thesis are all of this kind. One factor, initial number of users, is chosen and the other, operation stream sequences, is varied. The opposite extreme is a *full factorial design with replication* in which the system's response is measured for all possible input combinations. With this type of data, it is possible to determine the effects of all input variables and all of their interactions, along with an indication of the magnitude of an error. However, full factorial design can require a very large number of experiments. A system with m factors each of which has n possible levels requires m^n separate experiments. To reduce the total number of experiments that must be performed, the experimenter can reduce either the number of levels used for each input variable, or the total number of inputs.

3.6.1 Terminology

Researchers who focus on *causal relations* usually begin with an effect, and then search for its causes. The input variables or cause variable, or the one that identifies forces or conditions that act on something else, are called *factors*. In some disciplines the popular term is *independent variable*. These are the variables the experimenter can control or change.

The factors for the experiment conducted for this thesis are the *initial number of users*, *operation stream* and, in addition, for the multi-tree solution, the *height for the mini-trees*. The specific values to which a factor may be set is known as *levels* of the factor. For the experiment conducted for this thesis, the levels for the initial number of users is three, for operation stream, it is four and for the mini-tree height, it is seven.

The *response variable* is the output value that is measured as the input values are changed. It is the effect or the result or outcome of another variable and is commonly referred to in some disciplines as the *dependent variable*. Other terms used for it are *outcome variable* and *effect variable*.

This goal of this thesis is finding out how a broadcast encryption scheme implemented using a single binary tree, that shrinks and expands by level is response to population of users, compares in efficiency with the multi-tree solution proposed in (Chen, Ge, Zhang, Kurose, & Towsley, 2004) in terms of the following parameters:

- The *key storage cost at the member side,*
- The *multicast cost* and
- The *unicast cost*

Therefore these three are the *response variables* of the experiment.

Replication means completely rerunning the experiment with all of the same input levels. This is useful in situations where measurement of the response variable is subject to random variations so that replications of an experiment determine the impact of measurement error on the response variable. The experiment for this thesis deals purely with nonnegative[15] integers and therefore and therefore replication, whatever number of times is carried out, will give the same result. Indeed, the experiments are in a way just confirmatory tests that the model is working correctly.

An *interaction* between factors occurs when the effect of one factor depends on the level of another factor. In some disciplines, the variable whose level alters the impact of another is called a *moderating variable*. A moderating variable represents a process or a factor that alters the impact of an independent variable X on a dependent variable Y. It influences, or moderates, the relation between two other variables and thus produces an interaction effect. In other words, it has a strong contingent effect on the independent variable-dependent variable relationship.

[15] A nonnegative integer r, means that $r \geq 0$. Positive integer n, means that $n > 0$.

The following sections describe all the variables that were used in the experiments.

3.6.2 The response variables

As noted in previous section, the response variables are the performance metrics set out in the research question, *namely message expansion*, *user storage* and *unicast* i.e. outputs from the experiments are the performance metrics. The following sections are brief descriptions for each:

Multicast Cost or Message Length or Message Expansion

This is the length of the header that is attached to $F(K,S)$, which is proportional to m, the number of sets in the partition covering $U \setminus R$. The multicast cost is equal to the minimum number of subsets used to cover the active members in the key tree. This is the same as the number of maximal chains in the Steiner tree created by the key server. This is computed as explained at Section 2.5.1. This is also the same as the message header or message expansion and is a measure of the rekeying cost.

Storage size at the receiver

This is the I_u the private information the user needs to know in order to recover the TEK during rekeying. This has been shown in Section 2.5.2 to be

$$I_{\mathrm{u}} = \frac{1}{2}(h^2 + h) + 1$$

where h is the height of the key tree at the key server.
I_u could simply consists of all the keys k_i such that $u \in f_i$, or if the key assignment is more sophisticated it should allow the computation of all such keys.

This could also be interpreted as a measure of the time taken to process a message at the receiver; i.e. performing decryption and other types of operations such as a user determining if they are one of the intended receivers.

Unicast Cost

This is simply the unicast message count - the number of messages containing the private information (typically, keys) that a new member must be sent. Note that a new member being admitted or a member being revoked may cause other users to be sent their private information too.

For the experiment of the thesis, the unicast cost is the number of joining members and shifted members. The secret information is delivered to a user when that user joins the group for the first time. Since a member's position in the key tree is fixed, no additional unicast costs are incurred when the member returns to the group. But a returning user who has never been a leaf of the current key tree is really a shifted member and therefore also receives a unicast message.

3.6.3 The Factors

A factor is the presumed cause in an experimental study. It is the variable the experimenter has control over, what they can choose and manipulate i.e. the values of a factor are under experimenter control. It is usually what the experimenter thinks will affect the dependent variable. In this framework, the independent variables are a set of operations and the number of initial privileged users also referred to as members.

Number of initial Users

As part of the "booting process", the algorithms that implement the simulators read the users initially from a text file. The number of initial users is an input to the algorithm.

Operation streams

An operation stream is simply sequence of operations executed by the algorithm. A stream can be of any length i.e. any number of operations. The operations used to generate the results presented in Chapter 5 are the four operation which each lead to an increase or decrease of number of users or number of members.

operation	does this…
admit	introduces a new user as a member
delete	removes a user from the system
revoke	suspends a member, making the user a non member
restore	changes the status of a non-member to a member

An operation stream is a permutation of any number of these four operations. The single-tree algorithm does not need any other input. See figure 14.

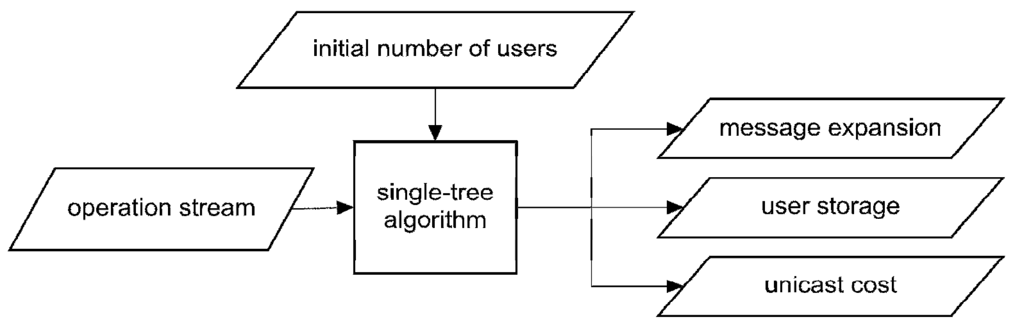


Figure 14: The single-tree solution

The mini-trees height

In the multi-tree solution, the height of the mini trees affect the user storage directly since the user storage is a measure of the tree height. This is because each user corresponds to a leaf in the key tree and the number of ancestors (which is a measure of the tree height) is a measure of the key user storage. Therefore the lower in the key tree a user is, the bigger the user storage. It may also affect the other variables. For example the bigger the height chosen, the bigger the key tree base and therefore the higher the chances of a poor user profile resulting in higher message expansion. Figure15 depicts the multi-tree solution setup.

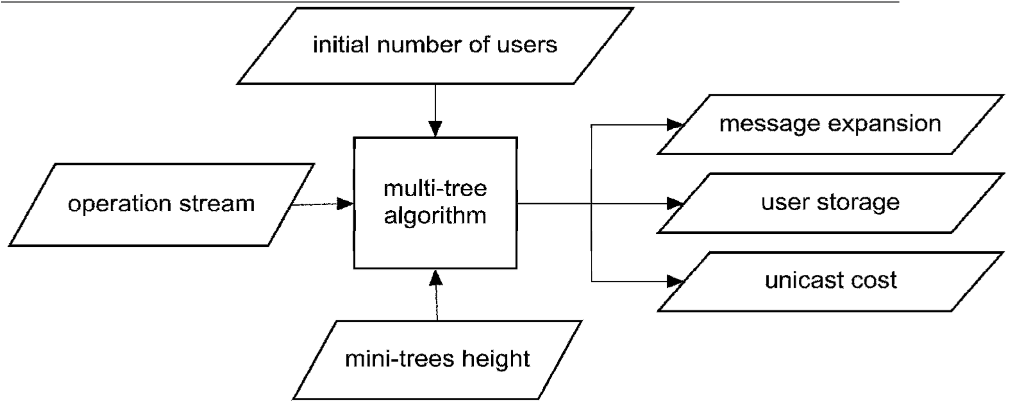


Figure 15: The Multi-tree solution

The following sections are a discussion on the experiment that produced the results presented in Chapter 4.

The experiments are carried out at two possible conditions; the typical condition and the cliff condition. The *typical condition* is the condition that the solution is assumed to operate is most of the time as justified in the explanation. The *cliff condition* is a condition in which the server may become unavailable for some time after an operation.

3.6.4 The cliff condition

The simulator uses a key tree. The tree used is a perfect binary tree. This means that the number of leaves of the tree is a power of two. In a key tree, a leaf corresponds to a user. The key tree space usage is therefore best when the number of members is a power of 2 i.e. 2^N. It should be clear that the worst case key tree space usage is when the number of members is one more than a power of 2 i.e. $2^N + 1$. These two extremes are referred to in this thesis as *cliff conditions* because either adding a member to 2^Nmembers or removing a member from $2^N + 1$ members results in tree recreation. During tree creation the server cannot process request because the key tree which is its most important component does not exist.

Worst-Case and Best-Case Efficiency

As explained in Chapter 3, the tree creation is the costliest operation in the single tree solution. When it takes place, the key server, G is not available to serve members. The efficiency of the single tree solution depends on how infrequently this operation takes place. This costly operation takes place because:

- a user is either being admitted as a member or
- a member is being revoked or
- a user is being deleted from the system or
- a previously revoked member is being readmitted.

Each of these operations is a complimentary operation to one of the other three operations. They are each applied three times in a complimentary manner to create an operation stream of 12 operations i.e. if *delete* is applied on user x, then the complementary *admit* is also applied on x in future. Each input stream of operations involves six users – three already in the system as members and three outside the system. The three users in the system are each revoked and reinstated as members while the users outside are each admitted as a member and deleted from the system.

The number of ways one can order the 12 operations is high. The orders that are of interest are only a few. For each of the two extreme possible space usage of the key tree, the way in which the operations are ordered (streamed) can results in the maximum possible times the key tree creation takes place - the *worst-case input operation stream* for the system. Similarly, there is the *best-case operation input.*

Definition: The worst-case efficiency of an algorithm is its efficiency for the worst-case input of size $\overline{\underline{n}}$ which is an input (or inputs) of size $\overline{\underline{n}}$ for which the algorithm runs the longest among all possible inputs of that size. (Levitin, October, 2011).

Definition: The best-case efficiency of an algorithm is its efficiency for the worst-case input of size $\overline{\underline{n}}$ which is an input (or inputs) of size $\overline{\underline{n}}$ for which the algorithm runs the fastest among all possible inputs of that size. (Levitin, October, 2011).

The worst-case key tree space usage is when the members are more than a power of two by one i.e. $2^N + 1$. For this number of users, the worst case operation input stream can be encoded WCWC where the first two characters correspond to operation and the last two for user space usage so that in full it is "worst-case operation stream for the worst-case user input". Similarly there is a BCWC ("best-case operation stream for the worst-case user input") stream. These two streams are summarized in $Table\ 3$.

Table 5: The streams used with worst initial user input

Serial	Operation	Tree Created?
1	revoke x	yes
2	restore x	yes
3	revoke y	yes
4	restore y	yes
5	revoke z	yes
6	admit u	yes
7	delete u	yes
8	admit v	yes
9	delete v	yes
10	admit w	yes
11	delete w	yes
12	restore z	yes

The WCWC stream

Serial	Operation	Tree Created?
1	admit u	no
2	admit v	no
3	admit w	no
4	delete w	no
5	delete v	no
6	delete u	no
7	revoke x	yes
8	revoke y	no
9	revoke z	no
10	restore z	no
11	restore y	no
12	restore x	yes

The BCWC stream

The best-case key tree space usage is when the members are a power of two i.e. 2^N. For this number of users, the worst case operation input stream will be known as BCWC and the best case operation input will be known as BCBC. These two operation inputs are summarized in *Table 4*.

Less or more than three applications of an operation in a stream could still work. For example one application of each operation could can work for the purpose of this experiment. However, the results presented are statistical – the mean value of the values obtained after each operation has been executed. In table 5, the expected response in terms key tree creation is listed for each operation. The worst-case operation input with one application of each of the four operations would still end up with the key tree being created two times. This might lead one to conclude that in the best case scenario, the tree creation (a very undesirable action) occurs two times within execution of four operations!

More than three applications of each operation lead to a harder means of determining the best-case sequence. Beyond three, may require an automated juggling of the operations to determine the best-case input – pencil and paper method becomes harder to use.

Table 6: The streams used with best initial user input

Serial	Operation	Tree Created?
1	`admit u`	yes
2	`revoke x`	yes
3	`restore x`	yes
4	`revoke y`	yes
5	`restore y`	yes
6	`revoke z`	yes
7	`restore z`	yes
8	`delete u`	yes
9	`admit v`	yes
10	`delete v`	yes
11	`admit w`	yes
12	`delete w`	yes

The WCBC stream

Serial	Operation	Tree Created?
1	`revoke x`	no
2	`restore x`	no
3	`revoke y`	no
4	`restore y`	no
5	`revoke z`	no
6	`restore z`	no
7	`admit u`	yes
8	`admit v`	no
9	`admit w`	no
10	`delete w`	no
11	`delete v`	no
12	`delete u`	yes

The BCBC stream

So there are four streams for the cliff condition

Stream name	Means…
WCWC	worst-case operation stream input for a worst-case key tree space usage
BCWC	best-case operation stream input for a worst-case key tree space usage
WCBC	worst-case operation stream input for a best-case key tree space usage
BCBC	best-case operation stream input for a best-case key tree space usage

From the tables above, we can see that the worst-case operation input is like a successful DoS attack on the key server. The system spends all the time creating the key tree and doing completely no useful work! But from the forgoing discussion, probability of this situation to occur is practice is almost nill. If it occurs at all, it is highly likely that it is an attack on the system. This could form a basis for an algorithm that detects if the system is under attack.

3.6.5 Typical Test Case

The four operation streams discussed in the previous section are meant for testing performance at *boundary condition* which in the context of this thesis is called *cliff conditions* tests. They are performed on a key tree that is either full completely or one that would be full if one member was not there. These two conditions are called the *cliff conditions* because either adding a member or removing a member results in tree recreation. In typical usage, the probability of reaching the cliff condition is slim from the following observation. If the current condition is cliff and members are

- increasing, then the next cliff arrives when members double

- reducing, then the next cliff arrives when members have reduced by half

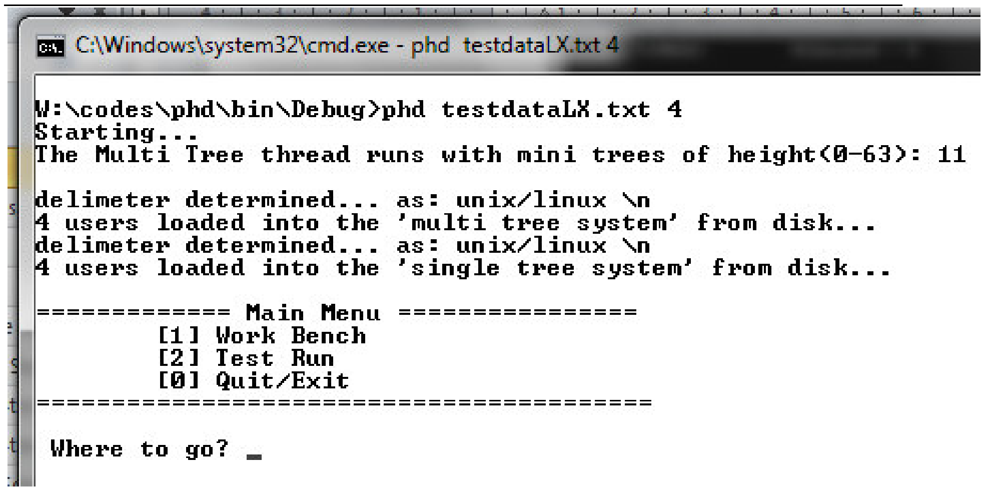
Therefore once the key tree crosses a *cliff condition*, the tree becomes stable for a “long time”. The only other time that the tree is recreated is when the holes are “too many” in the user profile. For this to occur, there should be admissions of new users and revocations of existing users only and these two be comparable in number. It is very unlikely in typical usage that revocations are comparable in number with admissions of new members.

Typical test case consists of a key tree with $1\frac{1}{2} \times 2^N$ members - this way the cliff condition is furthest and therefore the number of revocations and admissions are unlikely to result in key tree recreation. The operation streams used in the experiment are the same ones used for the cliff condition test.

3.7 How the simulator is run

The single-tree solution and the multi-tree solutions were simulated by a computer program written in C++. To ensure fairness for the two competing solutions, they are implemented using the same libraries and base classes. These are the steps one goes through when running the simulator:

- When one runs the simulator from the operating system command line, one may provide as arguments the source file that contains members’ names and the number of members to read into the system. If one does not provide these, the simulator reads users from a text file named “testdataLX.txt” and reads all the user names in the file. This default text file must reside in the same directory as the simulator executable file.
- Once the simulator starts running it asks for the height of the tree(s) the multi-tree solution will be using. This has to be known in advance and cannot change once the system has read it from the user.
- Then it reads the users into each of the implementations. But just before reading, it determines, among other things, the new line delimiter used. This differs between windows, Mac and unix-like systems.
- Once it has read users, it presents a menu of three items “Work Bench”, “Test Run” and “Exit”. Exit take the user out of the system back to the operating system. The other two lead to menus. See the Figure 16.

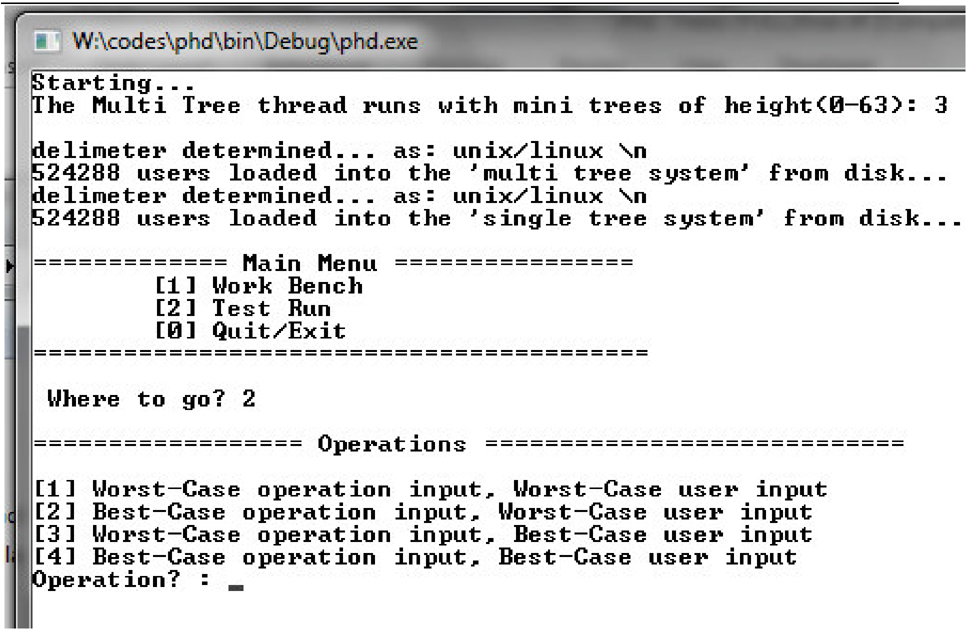


```
C:\Windows\system32\cmd.exe - phd  testdataLX.txt 4

W:\codes\phd\bin\Debug>phd testdataLX.txt 4
Starting...
The Multi Tree thread runs with mini trees of height(0-63): 11

delimeter determined... as: unix/linux \n
4 users loaded into the 'multi tree system' from disk...
delimeter determined... as: unix/linux \n
4 users loaded into the 'single tree system' from disk...

============= Main Menu ================
        [1] Work Bench
        [2] Test Run
        [0] Quit/Exit
=========================================

 Where to go? _
```

Figure 16: The simulator running with mini-trees height of 11 for the multi tree thread.

In this example, the users are initially 4; this value, 4 and the input file for users have been passed at the command line as arguments to the simulator executable.

If the user takes the "Test Run" branch, they are confronted with a menu of four choices; each a combination of "Best or Worst-case operation" and "Best or Worst-case user input". See the screen shot in figure 17.



```
W:\codes\phd\bin\Debug\phd.exe
Starting...
The Multi Tree thread runs with mini trees of height(0-63): 3

delimeter determined... as: unix/linux \n
524288 users loaded into the 'multi tree system' from disk...
delimeter determined... as: unix/linux \n
524288 users loaded into the 'single tree system' from disk...

============= Main Menu ================
        [1] Work Bench
        [2] Test Run
        [0] Quit/Exit
========================================

 Where to go? 2

================== Operations ============================

[1] Worst-Case operation input, Worst-Case user input
[2] Best-Case operation input, Worst-Case user input
[3] Worst-Case operation input, Best-Case user input
[4] Best-Case operation input, Best-Case user input
Operation? : _
```

Figure 17: The simulator waiting for user choice for test run.

For the cliff condition, the user should have read members totaling a power of two. Before it starts the test run that should be "Worst-case user input", it adds a user whose name is "omaya". Therefore for this test run to succeed there should be no user in the system with the name "omaya". The operation streams come from text files with the names "inop??user??.txt" where the "??" could be "bc" for best-case or "wc" for worst-case. Each of these files contain 12 lines that correspond to applications of the four operations "add", "delete", "revoke" and "restore" each applied three time. The first character on each line is a code for the operation as follows; 6 is revoke, 7 is restore, 8 is add and 9 is delete. A line that reads "7mangi" means restore back the previously revoked user whose user name is "mangi". Below is the actual content of "inopbcuserbc.txt" used in the test

```
========= content of "inopbcuserbc.txt" ==========

6makamba
7makamba
6shikanda
7shikanda
6abuyeka
7abuyeka
8mugabe
8uhuru
8opanga
9opanga
9uhuru
9mugabe
```

Once the user run one of these test runs, the output goes to a corresponding file named "outop??user??##.txt" where the "##" stands for the height that multi-tree uses in the test for the mini-trees. The content of this file is a detailed output which shows the state of affairs after each operation. It is quite long. The results that are tabled in, Chapter 4, Results, are manually gleaned from this output file. An actual output with 524,288 initial users for "outopbcuserbc11.txt" is shown in Appendix 2

The flowchart in Figure 18 shows how the simulator runs and interacts with the user.

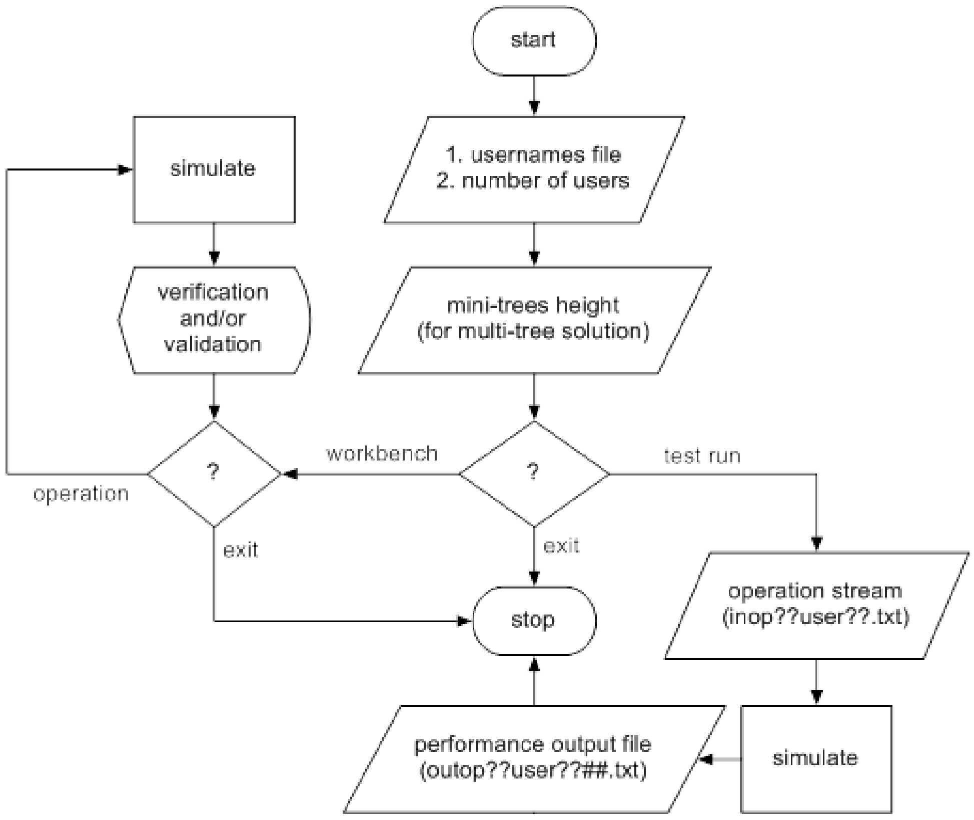


Figure 18: How the user interacts with the simulator

The summary results presented in the next chapter have been manually extracted from the performance output file generated by the simulator for presentation. The detailed output of the simulator has these summaries at the end of the file. As already noted, an actual output with 524,288 initial users for "outopbcuserbc11.txt" is shown in Appendix 2.

3.8 Note on the experiment

Note that to perform the tests, it is the user to choose the appropriate initial number of users.

- For the cliff condition test, the number of initial users should be a power of two i.e. 2^N or $2^N + 1$.
- For the typical condition test, a user must choose number of initial users that are half power of two plus a power of two i.e. $1\frac{1}{2} \times 2^N$

Chapter 4: Results and Discussion

The results are presented as a comparison of the performance of the multi-tree solution proposed in (Chen, Ge, Zhang, Kurose, & Towsley, 2004) and the proposed single-tree solution in this thesis using the metrics stated in section 1.8. Each of the operation streams described in the Section 3.6, Design of Experiment, for the single tree implementation is also applied on the implementation of the multi-tree Dynamic SDR running with allocations unit whose tree heights are 11, 15, 17, 19, 20, 21 and 22, and the "outputs" compared.

4.1 The cliff condition

The best case user space usage is attained when members are a power of 2 – members are equal to 2^n (n = 0, 1, 2, …, 63[16]). In such a case, the user profile has no holes and the message expansion is 1. On the other hand the worst case user space usage is when the members more than a power of 2 by one – members are equal to $2^n + 1$. For each of these two cliff conditions, there is a worst case operation stream input and a best-case operation stream input (see Section 3.6, Design of Experiment).

The summary of the *cliff condition test* results obtained in are shown in Table 7. For the boundary/cliff condition, the members (independent variable) is set at 2^{19} and $2^{19} + 1$. The former is the best case key tree usage and the latter the worst-case key tree usage. They are called cliff condition because of the potential "catastrophic" consequence of adding or removing a member. See the previous chapter for more details on this. A sample of the detailed results are in the Appendix 2, 3 and 4.

Generally, the taller the key tree, the bigger the storage on the user side. This is clear from the results of the multi-tree side. This is true also for the single-tree solution indeed for any solution based on a perfect binary (key) tree. Another point to note is that the shorter the tree, the lager the message expansion. The conservation laws roughly apply to these two metrics "when one increase the other must reduce and vice versa". This can be seen as one moves along the rows containing the values for the two parameters for the multi-tree solution.

[16] The C++ unsigned int data type limit

For the single tree solution, this can also be seen in the details (Appendix 2, 3 and 4) as one moves down the column for the single tree. Whenever the tree expands or shrinks the storage size decreases or increases respectively. The last column (result for the multi-tree) defy this trend though. The user storage goes higher but the message expansion is still stuck at 1. This is because of the weakness in the solution which is a relic of the static SD scheme. In static SD scheme, once a tree size has been chosen, it cannot change. This results in space abuse at the user side and even at the server. At user side because a user now must store more keys with one completely useless and at the server because a smaller tree could accommodate those users but now the tree has to as was chosen at initialization stage. As the single tree solution shown for the same number of users, they can fit on a smaller tree and the keys at the user can be less.

In the single-tree solution, the number of users determines the size of the tree, hence the key storage at the users. The tree size is the smallest possible, hence user storage is also the lowest possible. Therefore, whenever the multi-tree solution achieves a smaller key storage at the user side, it must perform poorer in message expansion.

Let us now look at the performance of the two at each performance metric. These results obtained confirm all the expectations:

Table 7 Cliff Condition mean results

		Single-Tree	Multi Tree (below is mini-tree height)						
			H=11	*H=15*	*H=17*	*H=19*	*H=20*	*H=21*	*H=22*
msg expansion	BCBC	1	256	16	4	1	1	1	1
	BCWC	1	257	17	5	2	2	2	2
	WCBC	1	256	16	4	1	1	1	1
	WCWC	1	257	17	5	2	2	2	2
unicast messages	BCBC	87381	0	0	0	0	0	0	0
	BCWC	87381	0	0	0	0	0	0	0
	WCBC	524288	0	0	0	0	0	0	0
	WCWC	524288	0	0	0	0	0	0	0
user storage	BCBC	199	67	121	154	191	211	232	254
	BCWC	202	67	121	154	191	211	232	254
	WCBC	201	67	121	154	191	211	232	254
	WCWC	201	67	121	154	191	211	232	254

4.2 Message Expansion

This is also known as Multicast Cost or Message Length. *Multicast cost* because you send the same message multiple times each encrypted with a different key and *Message Length* because from the server point of view, it is just a bunch of bytes being transmitted.

The high value for the multi-tree solution when the tree height is 0 is expected. Each user is in their tree alone. Therefore when sending a message, the message must be separately encrypted using each users key. The more users are packed in one tree, the smaller this value. This can be seen as bigger trees are used – as the reader move to the right across the tables.

The single tree-solution packs members in one tree all the time, and this tree is full (of users) on average throughout the 12 operations. That is why this is 1. If the reader looks at the details in the appendix, they notice that it becomes 2 at the seventh operation - when the tree expands.

The message expansion of the single tree solution is never higher than the message expansion of the multi-tree solution whatever the height for the mini-tree chosen. This is partly because of the design policy of the single tree solution of ensuring the user fit into the smallest possible tree and partly because of the design of the multi tree solution of potentially having members in multiple trees at the same time.

So the single tree solution is at worst as efficient as the multi-tree solution. This addresses the research question positively – the single tree solution is efficient in message expansion.

4.3 Storage Size at the Receiver

How much private information (typically, keys) does a receiver need to store? If the key assignment is more sophisticated the information should allow the computation of all the keys k_i such that $u \in \mathcal{f}_i$. For the results listed, I_u is obtained from the formula (see Section 2.5)

$$I_{\mathrm{u}} = \frac{1}{2}(h^2 + h) \ + \ 1$$

When the key tree is small, the key storage is small, because the size of the key is proportional to the height of the key tree. This can be seen as one moves from the left to the right – the tree that hosts a user becomes bigger as one moves from left to right.

At tree height of 19 for the multi-tree solution, the tree height is exactly the same size as the tree of the single tree implementation. That is why mean user storage are about equal. The small difference is caused by the fact that in the multi-tree solution, the user storage is constant while this changes in the single-tree solution.

Looking at the details in the appendix, one should notice that at the 7^{th} operation, the user storage goes up to 211 from 191. This is because the tree becomes bigger at this point.

The single tree solution ensures that user storage is always at its minimal. If we look at the results, we notice that the mean storage is higher for the single tree solution, at the boundary condition tests. This is because in these tests, the tree was created with a bigger height during the processing. For the multi-tree solution, another tree was created of the same size to accommodate the new user. So the multi-tree does not increase the user storage – it keeps it constant throughout – but the price it pays is a bigger space for the key tree at the server; the resulting two key trees of the multi-tree solution have more nodes in total than the single tree of the single-tree solution.

Otherwise, as we see later, during typical usage, the user storage remains stable for the single tree solution.

Storage size could also be interpreted as a measure of the time taken to process message at the receiver; i.e. performing decryption and other types of operations such as a user determining if they are one of the intended receivers.

Again here, the single tree solution answers the research question positively – it is efficient in storage at the user side. Indeed this is a positive consequence of the design goal of the single-tree solution. (See Section 2.5 for details)

4.4 Unicast Count

This is simply the unicast message count, the number of messages containing the private information (typically, keys) that a new member must be sent. This message must be sent to a member occupying a new location on the key tree – this could be because the member is new or they are shifting to new location from where they are now. Note that for the single-tree solution, a new member being admitted or a member being revoked may cause other users to be sent their private information too.

For the multi-tree solution, the unicast message count averages to zero

because it is only the admissions that require unicast. Out of the 12 operations, only three are admissions.

The unicast count at the boundary condition tests is higher for the single tree solution. This is expected because in each of the tests, key tree creation takes place in the course of executing the operations – two times for the BCWC and BCBC operation streams and 12 times for the WCBC and WCWC i.e. each time an operation is executed! When the key tree is created, a unicast must be sent to each user who will be hosted on it – all the current members. For the single tree solution, this means all members! As explained in section 3.6.4, the worst case operation input for either case results in the single tree system spending all its time creating the key tree. It is explained there that this is may indicate that the system is under a DoS attack since the sequence operation stream that should lead to the situation is "not natural".

For the multi-tree solution, an extra-tree is created and only users shifting to this newly created tree are sent a unicast. So in this metric, the single tree is potentially inefficient – in case a tree creation takes place.

4.5 Typical Usage Results

For typical usage, the value of N used is taken to be

$$N = 1\frac{1}{2} \times 2^{18}$$

This is a situation where the cliff condition is far – to reach a cliff, one third of the members need be removed or added. Therefore tree creation is unlikely and this is a typical situation in real practice. The same operation streams used for the boundary conditions are the same ones used with the hope that this can more or less depict typical usage.

The results shown next are the ones for a typical situation – far away from the boundary conditions.

Table 8: Typical mean results

		Single-Tree	Multi Tree						
			11	*15*	*17*	*19*	*20*	*21*	*22*
msg expansion	BCBC	1	192	12	3	1	1	1	1
	BCWC	2	193	13	4	2	2	2	2
	WCBC	2	192	12	3	2	2	2	2
	WCWC	2	193	13	4	2	2	2	2
unicast messages	BCBC	0	0	0	0	0	0	0	0
	BCWC	0	0	0	0	0	0	0	0
	WCBC	0	0	0	0	0	0	0	0
	WCWC	0	0	0	0	0	0	0	0
user storage	BCBC	191	67	121	154	191	211	232	254
	BCWC	191	67	121	154	191	211	232	254
	WCBC	191	67	121	154	191	211	232	254
	WCWC	191	67	121	154	191	211	232	254

In typical usage, the single-tree implementation is quiet stable;

- the message expansion is low and remains so
- the user storage is at its lowest possible and remains so
- the unicast cost is incurred only when a new user is admitted

4.6 Discussion

The design goal of the single key tree solution ensures that the user storage and message expansion are all the time at their possible lowest. At the boundary conditions, we see that this turns out to be catastrophic for the two worst-case operation inputs – the system spends all it time creating the key tree. In typical usage however, there is no chance that the tree creation takes place - these values remain at their lowest possible.

In typical usage, the two solutions tie on each metric when the multi-tree solution uses key tree whose height is 19. This is because the single-tree solution is also using a tree of same height. From the operations being used in the test, the tree does not get full for both or for the single-tree solution the members do not reduce by one third. When the tree is full, any addition of a member requires that the single-tree solution creates a new tree to accommodate them and for the multi-tree solution, it creates another tree of same size as the existing one to accommodate new members.

In typical usage, we see that in all the other columns where the multi-tree solution is not using a key trees with height of 19, it performs better on one metric but very poorly on another metric. When the height is less than 19, the message expansion is worse than the single tree's although the user storage is lower (therefore better). However remember that the header in a broadcast message (which is really message expansion) is the most important part when one analyses any broadcast encryption scheme. So really the single-tree is performing better on this metric.

At height of 19, they tie as pointed out but beyond height 19, they still tie but the user storage is higher for the multi-tree solution. So although they tie on the more important metric, the multi-tree solution does poorer on the user storage metric – and would do even worse with bigger heights because the bigger the key tree height, the higher the user storage.

Chapter 5: Conclusions and Recommendations

5.1 Trade-off between unicast cost and multicast cost

Dynamic SDR reduces multicast costs by inevitably introducing additional unicast, by which the secret information is delivered to shifted members or new members. In the single tree solution, emphasis is on holding users in the smallest key tree possible. A positive consequence is that the message expansion will always be low. This inevitably comes at a cost. The cost is an additional unicast over and above the multi-tree solution at the boundary conditions – when tree must be recreated because the current members are more than the existing tree can hold or they are few enough to be held on a smaller tree than the existing tree. As explained in Section 3.6.4, the probability of this occurring is very low.

5.2 Spreading the storage burden

The single tree solution is a more pure implementation of SD dynamic Revocation. The multi-tree solution still carries some of the baggage of the Static SD. In particular, once a tree size has been chosen, it remains so forever. To ensure that new users can be added, the multi-tree solution creates more trees of same height. This way, the burden on the user remains the constant throughout the operation of the system.

The single tree solution is more dynamic in the sense that even users take part in the implementation of the dynamism. When the single tree at the key server increases or reduces in size, each member feels the shock in the form of an increase or decrease respectively, in size of their key ring. This somewhat ensures the server and the members share the burden of space usage. For the same number of members, if they are hosted on multiple mini trees in the multi-tree solution, the total size of the key tree is larger than the single tree hosting the same number in the single tree solution.

There is only one disadvantage of the single tree solution over the multi-tree solution; the potential unicast to all members. This is very costly. However, as explained, this is an extremely rare event during typical usage service. Chances of this occurring are only real at the cliff condition – when the users are the tree is full (a power of 2) or a power of two plus 1.

This disadvantage is more than offset by the difficulty of making the right choice of the height, H for each tree in the multi-tree solution. The following is a quote from the Authors

> "The choosing of $\overline{L}$ ($\overline{L = 2^H}$) is a design tradeoff. Based on Proposition 1, one subset covers at most $\overline{L}$ members. Therefore, when $\overline{L}$ is small, more resultant subsets are required to cover $\overline{M}$. Consequently, the *multicast cost increases*. When $\overline{L}$ is large, we may still encounter the space inefficiency of static SDR that active members disperse in the subtree. We thus choose $\overline{L}$ as a reasonable value of $\overline{2^{\lceil \log E[M] \rceil}}$, where $\overline{E[M]}$ is the expected value of the number of concurrent members. Ideally, we want to put the concurrent members in one subtree." (Chen, Ge, Zhang, Kurose, & Towsley, 2004).

The last sentence in the quote is indeed true. When all the users are accommodated on a single tree (i.e. the two solutions use a tree of the same the same height), the multi-tree implementation beats the single tree solution in time usage and tie in all the other metrics. But knowing the maximum number of members in advance is difficult, a fact that the authors acknowledge in their abstract. The consequences of choosing the non-ideal height are clearly stated in the quote and is also explained in Section 4.3.

Another trouble with the multi-tree solution is the proposition that to improve the performance of the solution, the users on a mini-tree that have an ill formed user profile be migrated to the left-most tree and if this tree is full, a new tree created for them. This is a very costly event that involves among other things, a unicast to each migrating user.

5.3 Contribution

The three parameters set forth in the research question on which the solutions are compared are User storage, message expansion and unicast cost. Of these the most important is message expansion.

5.3.1 Typical usage

Message expansion

Message expansion is the number of times a message is encrypted and then broadcasted. In typical usage, the single tree solution's worst performance on message expansion is like the multi-tree's best. This is because the only time the multi-tree solution could match the single-solution is when all the users on the multi-tree are on one tree. More than one tree implies that the message expansion is at least two (one for each tree) while it may be one on the single-tree solution.

Unicast cost

The unicast cost is the number of messages to individual members. This is the message that conveys the new I_u (the new keyring) to a user when the user becomes a member. In typical usage, the unicast cost is like the multi-tree's. This is because in the single-tree solution, the tree is not recreated. A typical case is where the number of revocations and admissions of users is unlikely to reach the "cliff condition".

User Storage

The user storage may be equal or higher or lower than the multi-tree's.

- When it is higher, it means the multi-tree is more trees which are each shorter than the single tree being used by the single-tree solution. This translates to higher message expansion on the part of the multi-tree solution.
- When it is lower, it means the multi-tree is using one or more tree which are each taller than the single tree used by the single-tree solution. This also means that the multi-tree is using more space to store the key tree.

5.3.2 Cliff condition usage

Message expansion

Message expansion in cliff condition still remains minimal and the multi-tree can never do better than the single-tree solution. This is because all the time, the single-tree keeps users on the smallest possible tree that can accommodate them.

Unicast cost

This is the nightmare situation in the single-tree solution. Depending on the way the operations are arranged, it can result into the key server spending all its time recreating the key tree. As explained, this is a rare ly expected occurrence in practice. If it happens, the unicast cost is extremely high – whenever a tree is (re)created, a unicast message is sent to each member.

User storage

User storage is just like in the typical case and for exactly the same reasons

5.4 Suggestions for Future Work

5.4.1 The cliff condition nightmare

Some other algorithm can came into effect at the boundary condition that overrides the FIFO operation execution. Such an algorithm is worth designing. It may require one algorithm that detects when there is "progress towards a cliff" and another when there is "recession backwards to a cliff" and wakes up another to deal with the situation.

5.4.2 Assertions in this thesis

Another work that would be worth pursuing is a mathematical proof of each of the assertions in this thesis. Here are the assertions:

- A previously revoked member reoccupying their position in the binary tree on returning does not compromise forward secrecy. This contradicts (Chen, Ge, Zhang, Kurose, & Towsley, 2004) where they say "When a member, m, leaves the group, the position becomes empty and will never be used by any member (even m itself). And a new TEK is multicast to the members that remain in the group."
- The number of user cluster in the user profile directly affected the most important measure of performance of a BE, message expansion. This thesis attempts to show that the maximum number of clusters depends not only on the number of transitions (i.e. 01 or 10 subsequences in the user profile). A more rigorous mathematical proof is worth developing.

5.4.3 The computational time

This is caused by the fact that by design, any time a user leaves or is admitted, the single-tree implementation must determine if the users as they are after the operation (admission or revocation) need to be accommodated on a new tree to keep the storage cost at the user and message expansion at their lowest possible.

The computational class this algorithm belongs to could be worth studying.

References

Amaral, J. N. (2014, November 28). Retrieved November 28, 2014, from José Nelson Amaral's Publications: http://webdocs.cs.ualberta.ca/~c603/readings/research-methods.pdf

Anderson, K. (2005). *Tree Structures in Broadcast Encryption.* Linköpings universitet, SE-581 83 Linköping, Sweden, LIU-TEK-LIC-2005:70; Department of Electrical Engineering. Sweden: Linkoping Studies in Science and Technology, Thesis No. 1215.

Ayash, M. M. (2014, November 28). *Research Methodologies in Computer Science and Information Systems.* Retrieved November 28, 2014, from http://www.ptcdb.edu.ps/ar/sites/default/files/%D9%88%D8%B1%D9%82%D8%A9%20%D9%85%D9%87%D9%86%D8%AF%20%D8%B9%D9%8A%D8%A7%D8%B4.pdf

Bhattacherjee, S., & Sarkar, P. (June, 2012). Complete Tree Subset Difference Broadcast Encryption Scheme. *Designs, Codes and Cryptography. January 2013, Volume 66, Issue 1-3.*, 335-362.

Black, P. (Ed.). (2014, November 28). Retrieved November 28, 2014, from Dictionary of Algorithms and Data Structures: http://xlinux.nist.gov/dads/

Chen, W., & Dondeti, L. (October, 2002). Performance Comparison of Stateful and Stateless Gruop Rekeying Algorithms. *Fourth International Workshop on Networked Group Communication (NGC'02).* Boston, MA.

Chen, W., Ge, Z., Zhang, C., Kurose, J., & Towsley, D. (2004). On Dynamic Subset Difference Revocation Scheme.

Fiat, A., & Naor, M. (1994). Broadcast encryption. *CRYPTO '93 Proceedings of the 13th annual international cryptology conference on Advances in cryptology, ISBN:0-387-57766-1*, 480–491.

Johansson, M., Kreitz, G., & Lindholm, F. (2006). Stateful Subset Cover. *Applied Cryptography and Network Security Lecture, Notes in Computer Science, 3989*, 178-193.

Lassalle, D. (January 2005). Broadcast Encryption. *GIAC directory of certified professionals*.

Law, A. M. (January 2014). *Simulation Modeling and Analysis* (5th ed., Vols. ISBN-13: 978-0073401324). McGraw-Hill Series in Industrial Engineering and Management.

Levitin, A. V. (October, 2011). *Introduction to the Design & Analysis of Algorithms* (3rd ed., Vols. ISBN-13: 978-0132316811 ISBN-10: 0132316811).

Lilja, D. J. (2000). *Measuring Computer Performance, A Practitioner's Guide* (Vols. ISBN 0-521-64105-5). New York, NY: Cambridge University Press .

Lotspiech, J., Nusser, S., & Pestoni, F. (August 2002). Broadcast Encryption's Bright Future. *Computer, 35*(8), 57-63.

Naor, D., Naor, M., & Jeff, L. (July 2002). Revocation and Tracing Chemes for Stateless Receivers.

Obied, A. (April 2005). *Broadcast Encryption.* Calgary: Department of Computer Science, University of Calgary.

Wikipedia. (2014, November 28). *Broadcast encryption*. Retrieved November 28, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Broadcast_encryption

Zhang, Y.-C., Yang, C., Liu, J.-B., & Tian, J.-Y. (May 22-24, 2009). Broadcast Encryption Scheme and Its Implementation on Conditional Access System. *Proceedings of the 2009 International Symposium on Web Information Systems and Applications (WISA'09), ISBN 978-952-5726-00-8 (Print), 978-952-5726-01-5 (CD-ROM)*, pp. 379-382. Nanchang, P. R. China.

.

Appendix 1 (Actual Sample Output with WCWC stream)

The following is an actual output obtained for the cliff condition of 524,288 initial users and multi-tree implementation running with mini-tree of height 11. The operation stream is WCWC stream.

```
The <P8BESingle> starting...

Init time = <9221> clock ticks



OPERATION: <Revocation of makamba>

...Key Tree size after = <1048575> = nodes

...User space after = <524288> = slots

...Message Expansion = <1> = messages

...Users After = <524288> = users

...Users Storage = <191> = keys

...Unicast = <524288> = messages

..Took = <3081> = clock ticks

OPERATION: <Reinstatement of makamba>

...Key Tree size after = <2097151> = nodes

...User space after = <1048576> = slots

...Message Expansion = <2> = messages

...Users After = <524289> = users

...Users Storage = <211> = keys

...Unicast = <524289> = messages

..Took = <3419> = clock ticks

OPERATION: <Revocation of shikanda>

...Key Tree size after = <1048575> = nodes

...User space after = <524288> = slots

...Message Expansion = <1> = messages

...Users After = <524288> = users

...Users Storage = <191> = keys
```

```
...Unicast = <524288> = messages

..Took = <3019> = clock ticks

OPERATION: <Reinstatement of shikanda>

...Key Tree size after = <2097151> = nodes

...User space after = <1048576> = slots

...Message Expansion = <2> = messages

...Users After = <524289> = users

...Users Storage = <211> = keys

...Unicast = <524289> = messages

..Took = <3419> = clock ticks

OPERATION: <Revocation of abuyeka>

...Key Tree size after = <1048575> = nodes

...User space after = <524288> = slots

...Message Expansion = <1> = messages

...Users After = <524288> = users

...Users Storage = <191> = keys

...Unicast = <524288> = messages

..Took = <2971> = clock ticks

OPERATION: <Admission of mugabe>

...Key Tree size after = <2097151> = nodes

...User space after = <1048576> = slots

...Message Expansion = <2> = messages

...Users After = <524289> = users

...Users Storage = <211> = keys

...Unicast = <524289> = messages

..Took = <3421> = clock ticks

OPERATION: <Deletion of mugabe>

...Key Tree size after = <1048575> = nodes

...User space after = <524288> = slots

...Message Expansion = <1> = messages
```

```
...Users After = <524288> = users

...Users Storage = <191> = keys

...Unicast = <524288> = messages

..Took = <3008> = clock ticks

OPERATION: <Admission of uhuru>

...Key Tree size after = <2097151> = nodes

...User space after = <1048576> = slots

...Message Expansion = <2> = messages

...Users After = <524289> = users

...Users Storage = <211> = keys

...Unicast = <524289> = messages

..Took = <3420> = clock ticks

OPERATION: <Deletion of uhuru>

...Key Tree size after = <1048575> = nodes

...User space after = <524288> = slots

...Message Expansion = <1> = messages

...Users After = <524288> = users

...Users Storage = <191> = keys

...Unicast = <524288> = messages

..Took = <2981> = clock ticks

OPERATION: <Admission of opanga>

...Key Tree size after = <2097151> = nodes

...User space after = <1048576> = slots

...Message Expansion = <2> = messages

...Users After = <524289> = users

...Users Storage = <211> = keys

...Unicast = <524289> = messages

..Took = <3438> = clock ticks

OPERATION: <Deletion of opanga>

...Key Tree size after = <1048575> = nodes
```

```
...User space after = <524288> = slots

...Message Expansion = <1> = messages

...Users After = <524288> = users

...Users Storage = <191> = keys

...Unicast = <524288> = messages

..Took = <2976> = clock ticks

OPERATION: <Reinstatement of abuyeka>

...Key Tree size after = <2097151> = nodes

...User space after = <1048576> = slots

...Message Expansion = <2> = messages

...Users After = <524289> = users

...Users Storage = <211> = keys

...Unicast = <524289> = messages

..Took = <3421> = clock ticks

... <P8BESingle> finished...

   ...mean key tree size = <1572863> = nodes

   ...mean user storage = <201> = keys

   ...mean msg expansion = <1> = messages

   ...mean unicast messages = <524288> = messages

   ...Time taken = <38574> = clock ticks




The <11P7BEMulti> starting...

Init time = <9542> clock ticks


OPERATION: <Revocation of makamba>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages
```

```
...Users After = <524288> = users

...Users Storage = <67> = keys

...Unicast = <0> = messages

..Took = <0> = clock ticks

OPERATION: <Reinstatement of makamba>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524289> = users

...Users Storage = <67> = keys

...Unicast = <0> = messages

..Took = <0> = clock ticks

OPERATION: <Revocation of shikanda>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524288> = users

...Users Storage = <67> = keys

...Unicast = <0> = messages

..Took = <0> = clock ticks

OPERATION: <Reinstatement of shikanda>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524289> = users

...Users Storage = <67> = keys

...Unicast = <0> = messages

..Took = <0> = clock ticks

OPERATION: <Revocation of abuyeka>

...Key Tree size after = <1052415> = nodes
```

```
...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524288> = users

...Users Storage = <67> = keys

...Unicast = <0> = messages

..Took = <0> = clock ticks

OPERATION: <Admission of mugabe>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524289> = users

...Users Storage = <67> = keys

...Unicast = <1> = messages

..Took = <0> = clock ticks

OPERATION: <Deletion of mugabe>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524288> = users

...Users Storage = <67> = keys

...Unicast = <0> = messages

..Took = <1> = clock ticks

OPERATION: <Admission of uhuru>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <258> = messages

...Users After = <524289> = users

...Users Storage = <67> = keys

...Unicast = <1> = messages

..Took = <0> = clock ticks
```

```
OPERATION: <Deletion of uhuru>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524288> = users

...Users Storage = <67> = keys

...Unicast = <0> = messages

..Took = <1> = clock ticks

OPERATION: <Admission of opanga>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <258> = messages

...Users After = <524289> = users

...Users Storage = <67> = keys

...Unicast = <1> = messages

..Took = <0> = clock ticks

OPERATION: <Deletion of opanga>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524288> = users

...Users Storage = <67> = keys

...Unicast = <0> = messages

..Took = <1> = clock ticks

OPERATION: <Reinstatement of abuyeka>

...Key Tree size after = <1052415> = nodes

...User space after = <526336> = slots

...Message Expansion = <257> = messages

...Users After = <524289> = users

...Users Storage = <67> = keys
```

```
...Unicast = <0> = messages

..Took = <0> = clock ticks

... <11P7BEMulti> finished...

   ...mean key tree size = <1052415> = nodes

   ...mean user storage = <67> = keys

   ...mean msg expansion = <257> = messages

   ...mean unicast messages = <0> = messages

   ...Time taken = <3> = clock ticks

   ...a tree hieght = <12> levels
```

Appendix 2 (Sample Side-by-Side Output with WCWC stream)

The is the same output shown in Appendix 2 but here the outputs from the two solutions are shown side by side.

```
The <P8BESingle> starting...                      The <11P7BEMulti> starting...

Init time = <9221> clock ticks                    Init time = <9542> clock ticks



OPERATION: <Revocation of makamba>                OPERATION: <Revocation of makamba>

...Key Tree size after = <1048575> = nodes        ...Key Tree size after = <1052415> = nodes

...User space after = <524288> = slots            ...User space after = <526336> = slots

...Message Expansion = <1> = messages             ...Message Expansion = <257> = messages

...Users After = <524288> = users                 ...Users After = <524288> = users

...Users Storage = <191> = keys                   ...Users Storage = <67> = keys

...Unicast = <524288> = messages                  ...Unicast = <0> = messages

..Took = <3081> = clock ticks                     ..Took = <0> = clock ticks

OPERATION: <Reinstatement of makamba>             OPERATION: <Reinstatement of makamba>

...Key Tree size after = <2097151> = nodes        ...Key Tree size after = <1052415> = nodes

...User space after = <1048576> = slots           ...User space after = <526336> = slots

...Message Expansion = <2> = messages             ...Message Expansion = <257> = messages

...Users After = <524289> = users                 ...Users After = <524289> = users

...Users Storage = <211> = keys                   ...Users Storage = <67> = keys

...Unicast = <524289> = messages                  ...Unicast = <0> = messages

..Took = <3419> = clock ticks                     ..Took = <0> = clock ticks

OPERATION: <Revocation of shikanda>               OPERATION: <Revocation of shikanda>

...Key Tree size after = <1048575> = nodes        ...Key Tree size after = <1052415> = nodes

...User space after = <524288> = slots            ...User space after = <526336> = slots

...Message Expansion = <1> = messages             ...Message Expansion = <257> = messages
```

```
...Users After = <524288> = users                  ...Users After = <524288> = users
...Users Storage = <191> = keys                    ...Users Storage = <67> = keys
...Unicast = <524288> = messages                   ...Unicast = <0> = messages
..Took = <3019> = clock ticks                      ..Took = <0> = clock ticks
OPERATION: <Reinstatement of shikanda>             OPERATION: <Reinstatement of shikanda>
...Key Tree size after = <2097151> = nodes         ...Key Tree size after = <1052415> = nodes
...User space after = <1048576> = slots            ...User space after = <526336> = slots
...Message Expansion = <2> = messages              ...Message Expansion = <257> = messages
...Users After = <524289> = users                  ...Users After = <524289> = users
...Users Storage = <211> = keys                    ...Users Storage = <67> = keys
...Unicast = <524289> = messages                   ...Unicast = <0> = messages
..Took = <3419> = clock ticks                      ..Took = <0> = clock ticks
OPERATION: <Revocation of abuyeka>                 OPERATION: <Revocation of abuyeka>
...Key Tree size after = <1048575> = nodes         ...Key Tree size after = <1052415> = nodes
...User space after = <524288> = slots             ...User space after = <526336> = slots
...Message Expansion = <1> = messages              ...Message Expansion = <257> = messages
...Users After = <524288> = users                  ...Users After = <524288> = users
...Users Storage = <191> = keys                    ...Users Storage = <67> = keys
...Unicast = <524288> = messages                   ...Unicast = <0> = messages
..Took = <2971> = clock ticks                      ..Took = <0> = clock ticks
OPERATION: <Admission of mugabe>                   OPERATION: <Admission of mugabe>
...Key Tree size after = <2097151> = nodes         ...Key Tree size after = <1052415> = nodes
...User space after = <1048576> = slots            ...User space after = <526336> = slots
...Message Expansion = <2> = messages              ...Message Expansion = <257> = messages
...Users After = <524289> = users                  ...Users After = <524289> = users
...Users Storage = <211> = keys                    ...Users Storage = <67> = keys
...Unicast = <524289> = messages                   ...Unicast = <1> = messages
```

```
..Took = <3421> = clock ticks
OPERATION: <Deletion of mugabe>
...Key Tree size after = <1048575> = nodes
...User space after = <524288> = slots
...Message Expansion = <1> = messages
...Users After = <524288> = users
...Users Storage = <191> = keys
...Unicast = <524288> = messages
..Took = <3008> = clock ticks
OPERATION: <Admission of uhuru>
...Key Tree size after = <2097151> = nodes
...User space after = <1048576> = slots
...Message Expansion = <2> = messages
...Users After = <524289> = users
...Users Storage = <211> = keys
...Unicast = <524289> = messages
..Took = <3420> = clock ticks
OPERATION: <Deletion of uhuru>
...Key Tree size after = <1048575> = nodes
...User space after = <524288> = slots
...Message Expansion = <1> = messages
...Users After = <524288> = users
...Users Storage = <191> = keys
...Unicast = <524288> = messages
..Took = <2981> = clock ticks
OPERATION: <Admission of opanga>
...Key Tree size after = <2097151> = nodes
```

```
..Took = <0> = clock ticks
OPERATION: <Deletion of mugabe>
...Key Tree size after = <1052415> = nodes
...User space after = <526336> = slots
...Message Expansion = <257> = messages
...Users After = <524288> = users
...Users Storage = <67> = keys
...Unicast = <0> = messages
..Took = <1> = clock ticks
OPERATION: <Admission of uhuru>
...Key Tree size after = <1052415> = nodes
...User space after = <526336> = slots
...Message Expansion = <258> = messages
...Users After = <524289> = users
...Users Storage = <67> = keys
...Unicast = <1> = messages
..Took = <0> = clock ticks
OPERATION: <Deletion of uhuru>
...Key Tree size after = <1052415> = nodes
...User space after = <526336> = slots
...Message Expansion = <257> = messages
...Users After = <524288> = users
...Users Storage = <67> = keys
...Unicast = <0> = messages
..Took = <1> = clock ticks
OPERATION: <Admission of opanga>
...Key Tree size after = <1052415> = nodes
```

```
...User space after = <1048576> = slots
...Message Expansion = <2> = messages
...Users After = <524289> = users
...Users Storage = <211> = keys
...Unicast = <524289> = messages
..Took = <3438> = clock ticks
OPERATION: <Deletion of opanga>
...Key Tree size after = <1048575> = nodes
...User space after = <524288> = slots
...Message Expansion = <1> = messages
...Users After = <524288> = users
...Users Storage = <191> = keys
...Unicast = <524288> = messages
..Took = <2976> = clock ticks
OPERATION: <Reinstatement of abuyeka>
...Key Tree size after = <2097151> = nodes
...User space after = <1048576> = slots
...Message Expansion = <2> = messages
...Users After = <524289> = users
...Users Storage = <211> = keys
...Unicast = <524289> = messages
..Took = <3421> = clock ticks
... <P8BESingle> finished...
  ...mean key tree size = <1572863> = nodes
  ...mean user storage = <201> = keys
  ...mean msg expansion = <1> = messages
  ...mean unicast messages = <524288> = messages
```

```
...User space after = <526336> = slots
...Message Expansion = <258> = messages
...Users After = <524289> = users
...Users Storage = <67> = keys
...Unicast = <1> = messages
..Took = <0> = clock ticks
OPERATION: <Deletion of opanga>
...Key Tree size after = <1052415> = nodes
...User space after = <526336> = slots
...Message Expansion = <257> = messages
...Users After = <524288> = users
...Users Storage = <67> = keys
...Unicast = <0> = messages
..Took = <1> = clock ticks
OPERATION: <Reinstatement of abuyeka>
...Key Tree size after = <1052415> = nodes
...User space after = <526336> = slots
...Message Expansion = <257> = messages
...Users After = <524289> = users
...Users Storage = <67> = keys
...Unicast = <0> = messages
..Took = <0> = clock ticks
... <11P7BEMulti> finished...
  ...mean key tree size = <1052415> = nodes
  ...mean user storage = <67> = keys
  ...mean msg expansion = <257> = messages
  ...mean unicast messages = <0> = messages
```

```
...Time taken = <38574> = clock ticks
```

```
...Time taken = <3> = clock ticks

...a tree hieght = <12> levels
```

Appendix 3 (All Output with WCWC stream)

The following are values harvested from the outputs for the WCWC operation stream.

	Single tree	Multi-Tree						
		H=00	H=05	H=11	H=15	H=17	H=19	H=20
Init time (clock ticks)	9221	12401	9671	9542	9531	9530	9578	10838
Operations								
Revocation of makamba								
...Key Tree size after (nodes)	1048575	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	524288	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	1	524288	16385	257	17	5	2	2
...Users After(users)	524288	524288	524288	524288	524288	524288	524288	524288
...Users Storage(keys)	191	1	16	67	121	154	191	211
...Unicast(messages)	524288	0	0	0	0	0	0	0
..Took(clock ticks)	3081	0	0	0	0	0	0	1
Reinstatement of makamba								
...Key Tree size after (nodes)	2097151	524289	1032255	1052415	1114095	1310715	2097150	2097151

...User space after(slots)	1048576	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	2	524289	16385	257	17	5	2	2
...Users After(users)	524289	524289	524289	524289	524289	524289	524289	524289
...Users Storage(keys)	211	1	16	67	121	154	191	211
...Unicast(messages)	524289	1	0	0	0	0	0	0
..Took(clock ticks)	3419	0	0	0	0	0	0	0
Revocation of shikanda								
...Key Tree size after (nodes)	1048575	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	524288	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	1	524288	16385	257	17	5	2	2
...Users After(users)	524288	524288	524288	524288	524288	524288	524288	524288
...Users Storage(keys)	191	1	16	67	121	154	191	211
...Unicast(messages)	524288	0	0	0	0	0	0	0
..Took(clock ticks)	3019	0	0	0	0	0	0	0
Reinstatement of shikanda								
...Key Tree size after (nodes)	2097151	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	1048576	524289	524320	526336	557056	655360	1048576	1048576

...Message Expansion(messages)	2	524289	16385	257	17	5	2	2
...Users After(users)	524289	524289	524289	524289	524289	524289	524289	524289
...Users Storage(keys)	211	1	16	67	121	154	191	211
...Unicast(messages)	524289	1	0	0	0	0	0	0
..Took(clock ticks)	3419	0	0	0	0	0	0	0
Revocation of abuyeka								
...Key Tree size after (nodes)	1048575	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	524288	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	1	524288	16385	257	17	5	2	2
...Users After(users)	524288	524288	524288	524288	524288	524288	524288	524288
...Users Storage(keys)	191	1	16	67	121	154	191	211
...Unicast(messages)	524288	0	0	0	0	0	0	0
..Took(clock ticks)	2971	0	0	0	0	0	0	0
Admission of mugabe								
...Key Tree size after (nodes)	2097151	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	1048576	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	2	524289	16385	257	17	5	2	2

...Users After(users)	524289	524289	524289	524289	524289	524289	524289	524289
...Users Storage(keys)	211	1	16	67	121	154	191	211
...Unicast(messages)	524289	1	1	1	1	1	1	1
..Took(clock ticks)	3421	0	0	0	0	0	0	0
Deletion of mugabe								
...Key Tree size after (nodes)	1048575	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	524288	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	1	524288	16385	257	17	5	2	2
...Users After(users)	524288	524288	524288	524288	524288	524288	524288	524288
...Users Storage(keys)	191	1	16	67	121	154	191	211
...Unicast(messages)	524288	0	0	0	0	0	0	0
..Took(clock ticks)	3008	1	1	1	1	1	1	1
Admission of uhuru								
...Key Tree size after (nodes)	2097151	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	1048576	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	2	524289	16386	258	18	6	3	3
...Users After(users)	524289	524289	524289	524289	524289	524289	524289	524289

...Users Storage(keys)	211	1	16	67	121	154	191	211
...Unicast(messages)	524289	1	1	1	1	1	1	1
..Took(clock ticks)	3420	0	0	0	0	0	0	0
Deletion of uhuru								
...Key Tree size after (nodes)	1048575	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	524288	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	1	524288	16385	257	17	5	2	2
...Users After(users)	524288	524288	524288	524288	524288	524288	524288	524288
...Users Storage(keys)	191	1	16	67	121	154	191	211
...Unicast(messages)	524288	0	0	0	0	0	0	0
..Took(clock ticks)	2981	2	1	1	1	1	1	1
Admission of opanga								
...Key Tree size after (nodes)	2097151	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	1048576	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	2	524289	16386	258	18	6	3	3
...Users After(users)	524289	524289	524289	524289	524289	524289	524289	524289
...Users Storage(keys)	211	1	16	67	121	154	191	211

...Unicast(messages)	524289	1	1	1	1	1	1	1
..Took(clock ticks)	3438	0	0	0	0	0	0	0
Deletion of opanga								
...Key Tree size after (nodes)	1048575	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	524288	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	1	524288	16385	257	17	5	2	2
...Users After(users)	524288	524288	524288	524288	524288	524288	524288	524288
...Users Storage(keys)	191	1	16	67	121	154	191	211
...Unicast(messages)	524288	0	0	0	0	0	0	0
..Took(clock ticks)	2976	5	1	1	1	1	2	1
Reinstatement of abuyeka								
...Key Tree size after (nodes)	2097151	524289	1032255	1052415	1114095	1310715	2097150	2097151
...User space after(slots)	1048576	524289	524320	526336	557056	655360	1048576	1048576
...Message Expansion(messages)	2	524289	16385	257	17	5	2	2
...Users After(users)	524289	524289	524289	524289	524289	524289	524289	524289
...Users Storage(keys)	211	1	16	67	121	154	191	211
...Unicast(messages)	524289	1	0	0	0	0	0	0

..Took(clock ticks)	3421	0	0	0	0	0	0	0
Summary (means values)								
... key tree size(nodes)	1572863	524289	1032255	1052415	1114095	1310715	2097150	2097151
... user storage(keys)	201	1	16	67	121	154	191	211
... msg expansion(messages)	1	524288	16385	257	17	5	2	2
... unicast messages(messages)	524288	0	0	0	0	0	0	0
...Time taken(clock ticks)	38574	8	3	3	3	3	4	4