# A set theoretic approach to broadcast encryption

Thomas Martin

Technical Report
RHUL–MA–2005–5
5 April 2005


Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

# A SET THEORETIC APPROACH TO BROADCAST ENCRYPTION

Thomas Martin

Royal Holloway and Bedford New college,
University of London

*Thesis submitted to*
*The University of London*
*for the degree of*
*Doctor of Philosophy*
*2004.*

*To my namesake, Thomas "Tommy" Martin.*

# Abstract

Broadcast Encryption allows a centre to send information over a broadcast channel to a dynamically changing group of users. The performance is rated by the bandwidth required for the broadcast and the amount of secret information needed to be stored at the user end. It can also be rated by the computational overhead. In the "Stateless Receiver" model, receivers are incapable of storing any new information, or updating themselves, between broadcasts. We look at two Stateless Receiver schemes by Naor et al., the Complete Subtree Revocation Scheme and the Subset Difference Revocation Scheme. We improve the bound on the bandwidth for the Complete Subtree Revocation Scheme given by Naor from $t_{\max}(n,r) \leq r(k - \log_2(r))$ to $t_{\max}(n,r) = r(k-j) - 2(r - 2^j)$, where $j = \lfloor \log_2(r) \rfloor$. We prove a similar bound on the maximum bandwidth for the Subset Difference Revocation Scheme. We also derive formula for the average bandwidth for both schemes.

The schemes of Naor et al. are each based on a single binary tree. We construct some variations of the Complete Subtree Revocation Scheme, the first has more than one tree, the other is based on an $a$-ary tree. We calculate the improved performance in bandwidth (traded off against an increase in storage). We make meaningful comparisons between these schemes and existing ones. Finally, we show how to reduce the storage requirement of the Complete Subtree Revocation Scheme from $\mathcal{O}(\log_2(n))$ to a constant term.

# Acknowledgements

I would like to thank my supervisors Prof. Peter Wild and Dr. Keith Martin for their assistance, advice, support. I could not have completed this thesis without your insight and encouragement. I would also like to thank the staff and the other post-graduate students in the Mathematics Department for making my time spent at Royal Holloway pleasant and rewarding. I am grateful to Prof. Fred Piper, for taking a special interest in my well-being, and John Duffell (for not bringing up the original estimate of 6-8 weeks).

I am happy to acknowledge the financial support of the Engineering and Physical Science Research Council (EPSRC).

I want to thank my family for all their support and encouragement over the years: my Mother for her irreplaceable lasagna and optimism, my Father for the strength to get this far, and my army of siblings for never letting it go to my head. I owe a special debt of gratitude to Graham and Sally (and Sean and Sam) who, as well as helping me get started, gave me a home away from home.

And thanks to all my friends, especially Alex, Big G, Chris, Livi, Illana & James, Paula, Amy and the kids. Thank you Roger, for making me think of it as a challenge, and Laurence for watching my back ... literally for 4 years and actually for much longer.

And to the good people of Tesco, for making "Tesco Value Bourbon Creams".

And most importantly, Maura, for your unending patience and understanding.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Broadcast encryption is becoming increasingly important in commercial applications such as Pay Television, Digital Rights Management, as well as network security in general. The basic idea is that there is a broadcast centre with a large group of subscribers to a particular service. Naturally, we have an initialisation phase where secret keys are (securely) distributed. Later, the centre will have to communicate with some (but probably not all) of the users. Some users may need to be permanently revoked, others may only be temporarily revoked, i.e. they are not allowed to access certain material.

An application of this would be Pay-Per-View Television. The broadcaster would distribute set-top boxes to each of its customers (each containing some unique secret information). If the centre has some content to broadcast (a movie or live sporting event), then it must be done so only those customers who have paid for it can get access. Another example would be a on-line subscription service. Say this centre wants to send streaming audio content to its customers. Initially, each customer will be given the software to receive the broadcast as well as some secret information. Again, the centre must make the broadcast so that only allowed customers can access the content. Reasons for being disallowed or revoked might be a user failing to pay the subscription fee, re-broadcasting the content or revealing their secret information. Intuitively, the ratio of allowed users to revoked users would be very different in the two examples. In the Pay-Per-View case, we expect a small number of paying customers (it is unlikely most users want to watch most movies, especially

if there is more than one channel). Alternatively, the subscription service is likely to have many more allowed users. The fraction of users who sign up and then fail/forget to pay the subscription fee is likely to stay small.

There are various conditions and restraints that can be placed on both the centre and the users. We have already seen that the number of revoked users can vary. There may be limits on the memory of each user's storage device, or the bandwidth the centre has available. Any secret information may need to be stored in tamper-resistant hardware which, because of the cost would have to be limited in size. There may be an *a priori* bound on the number of users that would need to be denied access at any time. What the receiver is capable of can also place a limit on what is possible. There are the standard costs of computation and storage that have to be considered. Does the receiver have the processing power to decode the broadcast "on-the-fly"? If not, can the entire broadcast be stored for later processing?

One factor we will be looking at specifically is the constraint of "Stateless Receivers" [23]. In this model, receivers retain no memory of previous broadcasts. The only information needed by a receiver to decode a broadcast is that given to the receiver initially and the broadcast itself. In this situation we must rule out any re-keying protocol, which is a common technique in some broadcast methods. A re-keying protocol in a system with "Stateful Receivers" would involve somehow setting up a group key with all authorised users. Multiple broadcasts can be made with this key since it only has to be changed when any of the previously authorised users is to be denied access. Due to the nature of the Stateless Receivers, any group key set up for a broadcast can only be used for that broadcast. Even if the very next broadcast is destined for the exact same set of users, a new key has to be established as the previous one is forgot by the receiver. So there will be some repetition when it comes to multiple broadcasts to the same set of users. Consequently, Stateless Receivers are better suited for situations where the set of desired recipients is constantly changing in an erratic manner.

The motivation for the "Stateless Receiver" model is the desire for the

| **Stateful Receivers** | **Stateless Receivers** |
|---|---|
| Needs to be able to write to memory | Never needs to write to memory |
| May need to be constantly on-line | Can go off-line at will |
| Users can be permanently revoked | Users cannot be permanently revoked |
| Multiple broadcasts with the same key | Needs to establish a key for each broadcast |

Table 1.1: Comparison of Stateful and Stateless Receivers

hardware/software at the receiver end to be as simple as possible. For use in a decoder-box or a portable media player, not having to update existing data would mean a simpler device. If the receiver were a PC, then the ability of update local information would be less of a problem. However, if any secret information held on the device is long term and not subject to updating, then this allows for the use of tamper resistant hardware. Also, the Stateless model removes the need for the receiver to be "on-line" all the time. If keys needed to be regularly updated and a user misses an update, then they are lacking the information to access all future updates. The centre can allow for some "down-time" by repeating updates, adding to bandwidth costs. It could even allow the user to request updates, again at the cost of bandwidth and creating a bottleneck at the centre. But it is assumed that in Broadcast Encryption users only receive information. The downside of this is that there is no easy way to permanently remove a compromised user from the system. We will show that the bandwidth cost is to some extent dependent on the number of users that are to be revoked. Once it is apparent that a user's secret information is made public or stolen, that user will be added to the set of revoked users, and the bandwidth will be affected. Due to the Stateless nature of the receivers, this one user cannot be removed with a one-time operation. The effect that revoking them has will last for the lifetime of the system. The differences between Stateless and Stateful Receivers are summarised in Table 1.1.

The layout of the chapters is as follows. In the second chapter we define the concepts (Revocation Schemes, Broadcasts, centre, etc.) and notation

we will be using. We also describe the ways we measure the performance of the schemes, in particular formalising the existing notion of the maximum bandwidth by $t_{\max}(n, r)$ and introducing the measure of average bandwidth, $t_{aver}(n, r)$. As most Revocations Schemes have a graphical description, in this chapter we cover the basic Graph Theory we will be using.

In the third chapter we review previous work on Revocation Schemes, in particular the schemes of Naor et al. [23] and Asano [1]. We also summarise the work in the more general field of Broadcast Encryption.

In the fourth and sixth chapters we look at the two schemes by Naor et al.: the Complete Subtree Revocation Scheme and the Subset Difference Revocation Scheme. We improve existing analysis of $t_{\max}(n, r)$ for these schemes, as well as looking at the average bandwidth. We also describe a new method for reducing the storage required in the Complete Subtree Revocation Scheme.

In the fifth chapter we look at the different possible ways of combining and growing Revocation Schemes. This leads directly to the construction of the new Forest of Trees Revocation Schemes.

Finally, in the last chapter we compare the various schemes that have been reviewed/constructed. We describe all the steps necessary for a centre to find the optimal scheme for any particular needs or constraints.

# Chapter 2

# Definitions and Notation

In this chapter we will lay the groundwork for the discussion and analysis of Revocation Schemes. We will state what is required of a Revocation Scheme, define a Revocation Scheme as a mathematical object, and then give a protocol describing how this object can be used to achieve the desired results. We will define some measures of different properties of schemes that will be useful in comparisons. The final section of the chapter details some basic Graph Theory. Most of the schemes we will be looking at are derived from specific graphs known as Trees. Throughout this thesis we will follow the notation in [23] as much as possible.

We will first define the entities we will come across, before stating the problem. In terms of participants, we essentially just have a *broadcast centre* and *users*. The broadcast centre is the most active participant in this arena. It must have a one-way communication channel with the users, as well as needing to give initial secret information to each user at start-up. Periodically, the centre will want to deliver a *message* to the users, which it will do in the form of a *broadcast* or *transmission*. This broadcast is the encryption of a *message*, along with a *header* which is extra information needed for users to decrypt the message. The users need do nothing more than receive and process broadcasts.

The problem in its simplest terms is as follows: There is a broadcast centre with a large set of users $\mathcal{N}$ ($|\mathcal{N}| = n$). Each user will be given a set of keys (which we will call *establishment keys*). At regular intervals the centre will want to broadcast a (probably large) message. For any transmission the centre

wishes to make, the set of all users will be split into two sets. *Excluded/revoked users*, set $\mathcal{R}$ (of size $r$), that are to be denied access, and *privileged/authorized users*, set $\mathcal{P}$, to whom the content has to be delivered. All users fall into one of the two categories with respect to a given message and there is no overlap ($\mathcal{P} \cup \mathcal{R} = \mathcal{N}$, $\mathcal{P} \cap \mathcal{R} = \emptyset$). The centre will want to be able to broadcast the one message to everyone so that only users in $\mathcal{P}$ can use their establishment keys to get the message. Because we use the set $\mathcal{R}$ more often than the set $\mathcal{P}$, the broadcast algorithms we will be looking at are also known as *Revocation Schemes*. The algorithms that we use to generate the broadcast and the parameters we use to measure the performance usually depend on the set of revoked users as opposed to the privileged set, i.e. the formula will be written in terms of $r = |\mathcal{R}|$ not $|\mathcal{P}|$.

Before we give the mathematical definition of a Revocation Schemes, we need to define the following:

*Definition* 1. Given a collection of non-empty subsets $S = \{S_1, S_2, \ldots, S_\omega\}$, $S_j \subseteq \mathcal{N}$ for $j = 1, \ldots, \omega$, and a non-empty set $\mathcal{P} \subseteq \mathcal{N}$, a *Cover* of $\mathcal{P}$ is a set $\{S_{i_1}, S_{i_2}, \ldots, S_{i_t}\}$ for some subset $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, \omega\}$ such that:

$$\mathcal{P} = \bigcup_{j=1}^{t} S_{i_j},$$

and is called a *Disjoint Cover* if it has the added property that $S_{i_j} \cap S_{i_{j'}} = \emptyset$, $\forall j \neq j'$.

For our purposes, the set $\mathcal{P}$ will be the set of privileged users (sometimes referred to as $\mathcal{N} \setminus \mathcal{R}$), and the subsets $S_i$ are each defined to be the set of users who share a particular key. By forming a cover of $\mathcal{P}$ with these subsets, we will obtain the keys needed for the broadcast. The specifics of the broadcast will be described later. A cover for the empty set $\mathcal{P} = \emptyset$ is not defined. This is a trivial case for a broadcast (no privileged users) and will be dealt with separately. Whenever we talk about the *size* of a cover we mean the number of sets that make up the cover.

We can now begin to describe a *Revocation Scheme*. The design of the system can be divided into three parts.

1. The keys need to be generated and distributed to the users. This is a once-off initialisation, that is to be performed by the centre.

2. The centre needs an encryption algorithm that on input of any message $M$ and a set of users to be revoked $\mathcal{R}$, outputs a ciphertext to be broadcast, $B$. This will be used each time the centre needs to deliver a message to (some) users.

3. Each user needs a decryption algorithm that takes as input the user's secret keys and any broadcast $B$ and outputs the original message $M$ if and only if that user is not a part of $\mathcal{R}$.

There are three conditions that the system must satisfy:

- A broadcast can be constructed and sent so that any set of privileged users can decrypt the broadcast.
- No excluded user for that transmission can decrypt the broadcast.
- No adversary outside the system (i.e. not in the set of users $\mathcal{N}$) can decrypt any transmission.

Ideally, there are also some factors which we would like to limit:

- Each user should only have to store a "reasonable" number of keys.
- The broadcast that the centre has to make should not be prohibitively long.
- The user should not have to perform too many complex operations in order to obtain the message.

We use the following notation to discuss some of these constraints. We denote the establishment keys as $L_1, L_2, \ldots, L_\omega$ (the number of keys $\omega$ will vary from scheme to scheme). We label the users $u_1, \ldots, u_n$. For any user $u_i \in \mathcal{N}$, $U_i$ is the set of keys that user knows. We look to minimise the size of this set, i.e. minimise $|U|_{\max} = \max_i |U_i|$. In most of the schemes we will

look at, $|U_i|$ is constant for all users, so this maximum is often the same as the storage at any receiver. Also, recall that these keys do not change (because of the stateless nature of the receivers), so we expect these to be long-lived keys. We will use $t(\mathcal{N}, \mathcal{R})$ as a measure of the bandwidth, and we will define this explicitly once the broadcast algorithm has been clarified.

First we will give the basic definition of what a Revocation Scheme is, then we will describe a protocol to implement the scheme.

## 2.1 Revocation Scheme

*Definition* 2. A Revocation Scheme is a triple $(\mathcal{N}, \Omega, \gamma)$ where $\mathcal{N}$ is a set of users, $\mathcal{N} = \{u_1, u_2, \ldots, u_n\}$, $\Omega$ is an index set and $\gamma$ is a one-to-one function, $\gamma : \Omega \to 2^{\mathcal{N}}$, with the following conditions: For every non-empty $\mathcal{P} \subseteq \mathcal{N}$ there exists a subset $\{i_1, \ldots, i_t\} \subseteq \Omega$ such that $\{\gamma(i_1), \ldots, \gamma(i_t)\}$ is a cover of $\mathcal{P}$:

$$\mathcal{P} = \bigcup_{j=1}^{t} \gamma(i_j). \tag{2.1}$$

Also, $\gamma$ never maps onto the empty set: $\gamma(i) \neq \emptyset, \forall i \in \Omega$.

We have already discussed the set $\mathcal{N}$. The index set $\Omega$ is related to the set of all establishment keys. Each index will be a placeholder for a distinct key. For the most part, we will simply have that $\Omega = \{1, \ldots, \omega\}$. The function $\gamma$ assigns keys (or the indices of keys) to sets of users. In the description of the protocol we will see how these are combined to satisfy the requirements of the system.

## 2.2 Revocation Protocol

As we have already mentioned, there are three phases in the Revocation Protocol. Initialisation, Broadcast Encryption and Broadcast Decryption. We will assume that the centre has chosen a Revocation Scheme $(\mathcal{N}, \Omega, \gamma)$ satisfying (2.1). Moreover, it is not enough for a cover to always exist, the centre will need an efficient "cover algorithm" to find a cover for any subset.

### 2.2.1 Initialisation

The centre must randomly generate the secret establishment keys to be used, $\mathcal{L} = \{L_1, L_2, \ldots, L_\omega\}$, one for each of the indices in the set $\Omega$. However, before doing so we must specify the keyspace. The centre must choose two symmetric[1] encryption algorithms, $E^1$ and $E^2$. $E^1$ will encrypt the message with a one-time session key and $E^2$ will encrypt the session key with several establishment keys. Therefore, the distinct establishment keys must be chosen from the keyspace of $E^2$:

$$\begin{aligned} \mathcal{L} = \{L_1, L_2, \ldots, L_\omega\} &\in (KEYSPACE(E^2))^\omega \\ L_i \neq L_j &\forall \quad i \neq j. \end{aligned}$$

At the very least, the keyspace has to be big enough to ensure that there are enough keys for all the indices. The centre must also ensure that all users have the ability to decrypt with both of the algorithms (we call the decryption functions $D^1$ and $D^2$ respectively). The specific requirements of the encryption algorithms will be discussed in Section 2.2.4.

We use the following formula to show how $\gamma$ determines which establishment keys a user is given:

$$u_j \in \gamma(i) \Leftrightarrow L_i \in U_j.$$

The function $\gamma$ takes as input one of the indices $i$ and outputs the set of all users who are to be given the key $L_i$. The function is one-to-one, but not necessarily onto as that would require an extremely large set of secret keys $\mathcal{L}$. The function is not one-to-many since it is well-defined. We could allow the function to be many-to-one, but this would mean that we would have multiple keys serving the same purpose. There is no advantage in having more than one key shared among the same set of users. This would only result in the exact same scheme, only with users storing extra keys. We will restrict ourselves to looking at schemes with no redundant keys, hence $\gamma$ is one-to-one.

[1] Naor [23] gave an example of using an asymmetric algorithm for $E^2$, but that would only be needed if the Broadcaster was not the same as the key generating centre.

From $\gamma$, the centre can work out the opposite function:

$$\begin{aligned} \delta : \mathcal{N} &\rightarrow 2^{\Omega} \\ i \in \delta(u_j) &\Leftrightarrow L_i \in U_j. \end{aligned}$$

This function $\delta$ takes as input one of the users $u_j$ and outputs the set of all indices of keys that user $u_j$ is to be given. The centre needs to calculate $\delta(u_j)$ and securely deliver the resulting set of keys to $u_j$, for all users. The function $\delta$ is only used during the initialisation stage, whereas $\gamma$ will be used in each broadcast.

### 2.2.2 Broadcast Encryption

Once the initialisation phase is complete, each user has the keys they need, and the centre can begin broadcasting ($\gamma(i)$ is now the set of users who have the key $L_i$). As we said earlier, along with any message to be sent, $M$, there is an associated set of users who are revoked, $\mathcal{R}$. The centre must use the cover algorithm to form a cover of $\mathcal{N} \setminus \mathcal{R}$ using the sets $\gamma(i)$. Say the cover is $C = \{\gamma(i_1), \gamma(i_2), \ldots, \gamma(i_t)\}$, so that:

$$\mathcal{N} \setminus \mathcal{R} = \bigcup_{j=1}^{t} \gamma(i_j).$$

The centre generates a random session key $K$ for use with $E^1$. The message is encrypted with this key using $E^1$. The session key $K$ is encrypted with each key $L_{i_j}$ from the cover using $E^2$. Finally, we need to list the indices $i_j$ of each key used. This is so that any user can find which of his/her keys were used (if any) and on which encrypted block. So the actual broadcast would be:

$$B = < \underbrace{[i_1, i_2, \ldots, i_t, E^2_{L_{i_1}}(K), E^2_{L_{i_2}}(K), \ldots, E^2_{L_{i_t}}(K)]}_{header}, E^1_K(M) >,$$

where the superscript on $E$ defines which encryption algorithm is used, and the subscript defines the key.

The case where $\mathcal{P} = \emptyset$ (or equivalently $\mathcal{R} = \mathcal{N}, r = n$) is a special case. Since there are no privileged users, the centre does not make any broadcast at

all. The other extreme, $\mathcal{P} = \mathcal{N}$, will depend on the scheme. Most schemes will have some index $i$ such that $\gamma(i) = \mathcal{N}$, which means that there is one key that every user has. But whether the cover is comprised of one or many indices, the message will still have to be encrypted with a session key as described above, and not sent in the clear, since the message still needs to be protected from attackers outside $\mathcal{N}$.

### 2.2.3 Broadcast Decryption

On receipt of the broadcast $B$, a privileged user must do the following.

- Search through the labels $i_1, \ldots, i_t$ until he finds one corresponding to a key that he has, say $L_{i_j}$.
- Use this key to decrypt the corresponding encrypted session key, $D^2_{L_{i_j}}(E^2_{L_{i_j}}(K)) = K$.
- Use $K$ to decrypt the message, $D^1_K(E^1_K(M)) = M$.

A revoked user will not get past the first of these stages.

### 2.2.4 Encryption Functions

The message could have been encrypted with the establishment keys directly, removing the need for a session key and a second encryption function. But this would make the length of the broadcast equal to the length of the message times the size of the cover (and that's assuming no message expansion). As it is described above, we only add a header that is roughly the size of the cover times the size of an $E^1$ key, which is likely to be much smaller than the message. Also, if the establishment keys were used directly, they would be vulnerable. If an attacker had a large quantity of ciphertext (which we have said is very likely), he may be able to cryptanalyse it to obtain information about the key. The establishment keys are long-lived, and must be kept secret.

The two encryption functions serve very different purposes and have different requirements. $E^1$ is applied to the message, so must be suitable for large

amounts of plaintext/ciphertext. Namely, it needs to be fast and not expand the plaintext. Also, extremely high security is not essential for $E^1$. Only session keys are used, and they can be changed for each broadcast. They need to be re-broadcast with each message anyway, because the stateless receivers do not store received messages. So attacking $E^1$ will only provide an adversary with at most the one-time session key and the message. It may even be the case that the message has a limited useful life, for example a live sporting event, so taking the time to break $E^1$ may mean that the message no longer has any value. $E^2$, on the other hand, uses the establishment keys $L_i$, which in a stateless receiver are long-lived. This means that we need to keep them secret for as long as possible. It is used on very short messages, namely keys from the keyspace of $E^1$. In some sense this gives us the opposite requirements to $E^1$, in that we want a strong cipher, but that speed is of less importance. The obvious solution to these specifications is to use a respected stream cipher for $E^1$ and a strong block cipher for $E^2$.

In generating a broadcast, the centre uses the cover algorithm to find a cover of all privileged users. In their paper describing the Complete Subtree and Subset Difference Revocation Schemes [23], Naor et al. also give a "Traitor Tracing" algorithm that requires (amongst other things) that the only cover used in the Revocation Protocol is a disjoint cover. If a group of users decide to pool their secret knowledge to form a pirate decoder, then Traitor Tracing is a method to use the pooled information to find the identity of at least one of the traitors ([9]). We will show that in most cases there is no advantage to be gained in allowing covers with overlapping sets.

### 2.2.5 Basic Efficiency Bounds

The description of the Revocation Protocol described the contents of the header of a broadcast, namely a list of indices and the encryption of the session key under the corresponding establishment keys. There are a few factors that determine the length of broadcast header. The size of the key $K$ and any message expansion from encryption with $E^2$ will contribute. But the

most important factor is the size of the cover. In any specific instance of a broadcast, we use the following notation to represent the size of the cover:

*Definition* 3. Let $(\mathcal{N}, \Omega, \gamma)$ be a Revocation Scheme. For any $\mathcal{R} \subset \mathcal{N}$ we denote by $t(\mathcal{N}, \mathcal{R})$ the minimal number of sets in any cover of $\mathcal{N} \setminus \mathcal{R}$.

If it is not clear from the context what Revocation Scheme was used to find the cover then it will be stated explicitly, i.e. $t^{RS}(\mathcal{N}, \mathcal{R})$ is the size of the minimal cover of $\mathcal{N} \setminus \mathcal{R}$ in the Revocation Scheme $RS$. The size of the cover, and consequently of the header, is the main source of the communication costs in the scheme (aside from the encrypted message, which is unavoidable). Therefore, we will use $t(\mathcal{N}, \mathcal{R})$ to measure the required bandwidth of any broadcast algorithm that we discuss, looking at both the maximum and the average value of the parameter $t(\mathcal{N}, \mathcal{R})$. Clearly this cost will depend on the size of the population of users, but it is also related to the number of revoked users. Therefore the two functions we will be interested in are:

$$t_{\max}(n, r) = \max_{\substack{\mathcal{R} \subseteq \mathcal{N} \\ |\mathcal{R}|=r}} (t(\mathcal{N}, \mathcal{R})), \tag{2.2}$$

$$t_{aver}(n, r) = \sum_{\substack{\mathcal{R} \subseteq \mathcal{N} \\ |\mathcal{R}|=r}} \frac{t(\mathcal{N}, \mathcal{R})}{\binom{n}{r}}. \tag{2.3}$$

The first function, $t_{\max}(n, r)$ is a formalisation of the standard measure of the bandwidth of a Revocation Schemes in the existing literature. The second function, $t_{aver}(n, r)$, the expected length of the broadcast header, has received much less attention. There has only been one scheme for which the average bandwidth has been investigated, the Subset Difference Revocation Scheme in [23], and that was just an upper bound. This is more likely due to practicality rather than the merit of $t_{aver}(n, r)$ compared to $t_{\max}(n, r)$. It is a lot more difficult to make statements about the average header length than the maximum header length. When looking at the bandwidth of several schemes, superscripts will be used to identify which scheme was used to calculate $t(\mathcal{N}, \mathcal{R})$, i.e. $t^{RS}_{\max}(n, r)$, $t^{RS}_{aver}(n, r)$.

One implicit assumption in the definition of a Revocation Scheme, is that we can always find a cover of the privileged users, for any choice of $\mathcal{R}$. Luckily,

we can place a simple constraint on $\Omega$ and the function $\gamma$ which will guarantee that this is always possible. All we need is for each user to have one key that they share exclusively with the centre.

*Theorem* 4. Let $\mathcal{N} = \{u_1, \ldots, u_n\}$, $\Omega = \{1, \ldots, \omega\}$ and $\gamma : \Omega \to 2^{\mathcal{N}}$. Define $S = \{\gamma(1), \ldots, \gamma(\omega)\}$. Then $(\mathcal{N}, \Omega, \gamma)$ is a Revocation Scheme if and only if

$$\{u\} \in S, \text{ for all } u \in \mathcal{N}.$$

*Proof.* Assume that $\{u\} \in S$, for all $u \in \mathcal{N}$. Therefore there exists $n$ indices $i_1, \ldots, i_n$ such that $\gamma(i_j) = \{u_j\}$. For simplicity sake we will relabel the indices so that $\gamma(i) = \{u_i\}$ for $i = 1, \ldots, n$. Given any set $\mathcal{P}(\subseteq \mathcal{N}) = \{u_{i_1}, \ldots, u_{i_p}\}$ we can define a cover to be:

$$\begin{aligned} \text{Cover} &= \{\gamma(i_1), \gamma(i_2), \ldots, \gamma(i_p)\}. \\ \text{Indeed} \quad \bigcup_{j=1}^{p} \gamma(i_j) &= \{u_{i_1}, u_{i_2}, \ldots, u_{i_p}\} \\ &= \mathcal{P}, \end{aligned}$$

which makes it a valid cover satisfying (2.1).

Conversely, suppose $(\mathcal{N}, \Omega, \gamma)$ is a Revocation Scheme. Then there exists a cover for any set. Let $u \in \mathcal{N}$. From the requirement (2.1) of a Revocation Scheme, we can always find a cover of the privileged set. So there exists a set of indices $\{i_1, \ldots, i_t\} \subseteq \Omega$ such that

$$\{u\} = \bigcup_{j=1}^{t} \gamma(i_j).$$

The sets $\gamma(i_j)$ are always distinct since $\gamma$ is one-to-one, and non-empty since $\gamma$ never maps to the empty set. It follows that the size of the cover is one and $\gamma(i_1) = \{u\}$. Thus $\{u\} \in S$ for all $u \in \mathcal{N}$. □

This theorem gives us the only necessary condition for a Revocation Scheme. We now try to minimise the costs of system, like the length of the broadcast header ($t(\mathcal{N}, \mathcal{R})$) and the amount of storage required at the user's end ($|U|_{\max}$). The following lemmas show how these factors can be minimised independently.

*Lemma* 5. Let $(\mathcal{N}, \Omega, \gamma)$ be a Revocation Scheme with $|U|_{\max} = 1$. Then $t_{\max}(n, r) = n - r$, for all $r = 0, \ldots, n-1$ (where $|\mathcal{N}| = n$).

*Proof.* We showed in Theorem 4 that every user $u$ must have one key to itself, namely $L_i$ where $\gamma(i) = \{u\}$. That means that $|U| \geq 1$. But if $|U|_{\max} = 1$ then $|U| = 1$ for all $u \in \mathcal{N}$. Since no key is shared by more than one user, this means $|\gamma(i)| = 1$, for all establishment keys $L_i$.

Given a message and a revoked set $\mathcal{R} \neq \mathcal{N}$, $|\mathcal{R}| = r$, we have to form a cover of $\mathcal{N} \setminus \mathcal{R}$. We can only use sets of size 1, which means we need at least $n-r$ sets (cardinality of $\mathcal{N} \setminus \mathcal{R}$). Since we are not allowed to repeat sets or use the null set, this means that the size of the cover, $t(\mathcal{N}, \mathcal{R})$, is always exactly $n - r$. Therefore $t_{\max}(n, r) = n - r$. □

Because the keys in Lemma 5 are required to be present (by Theorem 4), we can use these keys to form a cover as described above in any Revocation Scheme.

*Corollary* 6. Let $(\mathcal{N}, \Omega, \gamma)$ be a Revocation Scheme. Then

$$t_{\max}(n, r) \leq n - r.$$

*Proof.* For subset $\mathcal{R} \subseteq \mathcal{N}$ with $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$, we can cover $\mathcal{N} \setminus \mathcal{R}$ with the $n - r$ subsets as described in Lemma 5. If this is the minimal cover then $t_{\max}(n, r) = n - r$. Otherwise, there is a smaller cover and $t_{\max}(n, r) < n - r$. Therefore:

$$t_{\max}(n, r) \leq n - r.$$ □

We now minimise the bandwidth:

*Lemma* 7. Let $(\mathcal{N}, \Omega, \gamma)$ be a Revocation Scheme with $t_{\max}(n, r) = 1$, for all $r = 0, \ldots, n-1$. Then $|U|_{\max} = 2^{n-1}$.

*Proof.* In the given range for $r$, the number of privileged users, $n - r$, is always at least one. Whenever we have a non-zero number of privileged users to be covered ($n - r, r \neq n$), $t(\mathcal{N}, \mathcal{R})$ will have to be at least 1. So we have a Revocation Scheme where $t(\mathcal{N}, \mathcal{R})$ is always 1, except for the trivial case

where $\mathcal{P} = \emptyset$. As $t(\mathcal{N}, \mathcal{R}) = 1$ we can form any possible cover using just one set of the form $\gamma(i)$. So every non-empty subset of $\mathcal{N}$ has an associated index in $\Omega$. For any user $u \in \mathcal{N}$, $\delta(u)$ contains an index from $\Omega$ for every subset of $\mathcal{N}$ that contains $u$. Since $|\mathcal{N}| = n$ and $\gamma$ is one-to-one this means $|\delta(u)| = 2^{n-1}$. But $\delta(u)$ is just the set of keys to be given to user $u$. Therefore, $|U|_{\max} = 2^{n-1}$. □

These two lemmas show the only cases where we can get either the value of $t_{\max}(n, r)$ or the value of $|U|_{\max}$ down to 1, and neither is very practical. The parameters can be minimised individually, but you can not minimise them simultaneously. However, the above examples do give some idea of the trade-off involved between storage and bandwidth. We hope to find the relationship between these two parameters and explore the best "middle ground".

Theorem 4 allows us to say the following about $t_{\max}(n, r)$.

*Lemma* 8. For any Revocation Scheme

$$t_{\max}(n, r) \geq t_{\max}(n, r-1) - 1 \quad \text{for } r = 0, \ldots, n-1.$$

*Proof.* Set $t_{\max}(n, r) = t_1$. This means that for any subset $\mathcal{R} \subseteq \mathcal{N}$, $|\mathcal{R}| = r$, we have that $t(\mathcal{N}, \mathcal{R}) \leq t_1$. Consider any subset $\mathcal{R}'$, of size one less, $|\mathcal{R}'| = r - 1$. Now we wish to cover $\mathcal{N} \setminus \mathcal{R}'$. Let $u \in \mathcal{N} \setminus \mathcal{R}'$. Then $|\mathcal{R} \cup \{u\}| = r$ and there exists a cover of $\mathcal{N} \setminus \{\mathcal{R}' \cup \{u\}\}$ using at most $t_1$ subsets. Now these subsets together with $\{u\}$ form a cover of $\mathcal{N} \setminus \mathcal{R}'$. Therefore:

$$t(\mathcal{N}, \mathcal{R}') \leq t_1 + 1.$$

Since this is true for all $\mathcal{R}' \subseteq \mathcal{N}$ we have that $t_{\max}(n, r-1) \leq t_1 + 1$. This gives:

$$t_{\max}(n, r-1) \leq t_{\max}(n, r) + 1 \quad \text{i.e.} \quad t_{\max}(n, r) \geq t_{\max}(n, r-1) - 1. \qquad \square$$

This means that $t_{\max}(n, r)$ can only ever decrease by at most one as $r$ increases by one. Unfortunately, it does not limit how much $t_{\max}(n, r)$ can increase.

## 2.3 Graph Theory

The two Revocation Schemes, "Complete Subtree" and "Subset Difference", both use binary trees in their description as well as in discussions, bounds and proofs. We will first review the basic definitions:

A *graph* is set of *nodes* (or points, or vertices), some of which are joined by *edges*. Each edge is a pair of nodes (i.e. edge $e = (u, v)$ connects $u$ to $v$). In a *directed graph* or *digraph*, the pair is ordered, the edge starts at one node and ends at the other. We can talk about the *node set* $V(G)$ and the *edge set* $E(G)$ of a graph $G$, but mostly we will just say nodes and edges, respectively. The *degree* of a node is the number of edges connected to that node. A *path* in a graph $G$ is a sequence $v_1, e_1, v_2, e_2, \ldots, e_i, v_{i+1}$ where $v_i \in V(G)$ and $e_i \in E(G)$ are all distinct, and the edge $e_i$ connects $v_i$ and $v_{i+1}$. A path is a way to get from one node to another only using edges in $G$, without repeating nodes or edges. The *length* of a path is the number of edges in the sequence that defines the path. A *connected* graph is a graph where there is at least one path between every pair of nodes. A *cyclic* graph is a graph where there exists a path in the form $v_1, e_1, v_2, e_2, \ldots, v_i, e_i, v_1$ (a path that finishes where it starts). A *loop* is an edge that connects a node to itself ($e = (v, v)$). A *subgraph* $G'$ of a graph $G$ is a graph with $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$ and each edge in $E(G')$ connects a pair of nodes from $V(G')$.

### 2.3.1 Binary Trees

A *tree* is a undirected (not a digraph), connected, acyclic graph with no loops or multiple edges (edges that connect the same pair of nodes). Since it is connected, there is a path between every pair of nodes, and because there are no loops, multiple edges or cycles, this path is unique. Any node in a tree can be chosen to be the *root* of the tree, with all other nodes and edges of the graph being drawn descending from it (this is a *rooted tree*). Each node $v$, except the root, has a unique *parent* node, which is the node connected to $v$ on the path between $v$ and the root. All other nodes connected to $v$ are called its *child* nodes (the child nodes of the root are any nodes connected to it).

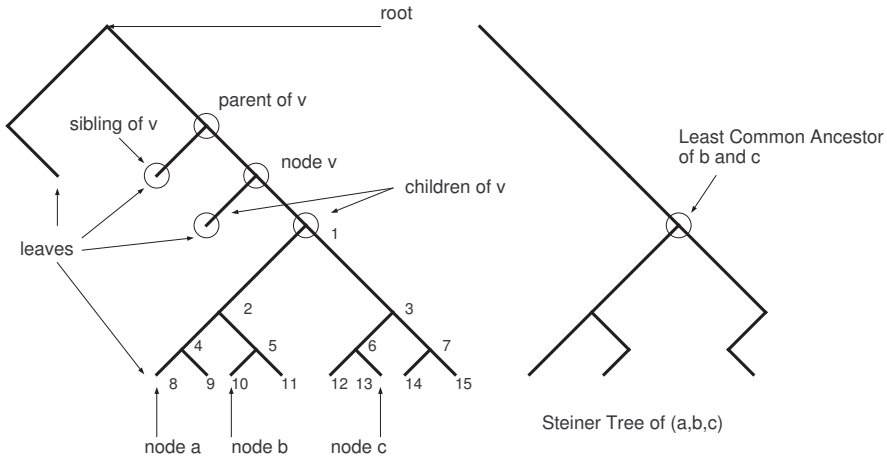


Figure 2.1: Binary tree notation.

The children of a node's parent are called *siblings* of the node. In formulae, we will use $par(v)$, $left\_child(v)$, $right\_child(v)$ and $sib(v)$ to refer to the parent, left child, right child and sibling of $v$ respectively.

The nodes of degree 1 that do not have child nodes themselves are called *leaves* (these will occur at the bottom of the tree if it is drawn with the root at the top). A node that is neither the root nor a leaf is called an *internal node*. The set of *descendants* of a node $v$ is the set of all leaves whose paths to the root pass through $v$. The leaves in this set are said to be *descended* from $v$, and $v$ is said to be the *ancestor* of the leaves. Parents are typically drawn above the node and children below, as in Figure 2.1. We measure the distance between two nodes as the number of edges on the path between them. So a parent of a node is closer to the root than its child.

A *binary tree* is a rooted tree where any node can have up to two child nodes. The obvious generalisation of this is an *a-ary tree*, in which each node can have up to $a$ child nodes. For the most part we will be working with binary trees. The *depth* of a binary tree is the maximum of the distances from the root to a leaf. We can also talk about the *depth* of a particular node, which is the distance from that node to the root, and the *height* of a particular node, which is the maximum of the distances from that node to a leaf that is

descended from it. A *complete binary tree* is a binary tree where all leaves are the same distance from the root and all internal nodes have degree 3 (and the root has degree 2). The number of leaves in a complete binary tree that has depth $k$ ($k$ edges from root to leaf) is $n = 2^k$ (which we will sometimes refer to as the *tree length*). No binary tree of the same depth can have more leaves than a complete binary tree as all nodes have the maximum possible degree. As a result of this we have:

*Lemma* 9. Any binary tree with $r$ leaves has depth at least $\lceil \log_2(r) \rceil$.

*Proof.* First, we need to show that $\lceil a \rceil < a+1$, for all $a \in \mathbb{R}$. If $a$ is an integer, then this is clearly true as $\lceil a \rceil = a$. Otherwise, $a = a' + \epsilon$, where $0 < \epsilon < 1$ and $a' \in \mathbb{Z}$. So $\lceil a \rceil = a' + 1$ and so:

$$\begin{aligned} a &= a' + \epsilon \\ a &> a' && \text{since } \epsilon > 0 \\ a + 1 &> a' + 1 && \text{adding 1 to the above} \\ a + 1 &> \lceil a \rceil && \text{since } \lceil a \rceil = a' + 1. \end{aligned}$$

Proof by Contradiction. Say tree $T$ has $r$ leaves and depth $\leq \lceil \log_2(r) \rceil - 1$. A complete binary tree with this depth would have $2^{\lceil \log_2(r) \rceil - 1}$. As this is the most leaves possible for a tree with this depth, an upper bound for the number of leaves in $T$ is:

$$2^{\lceil \log_2(r) \rceil - 1} < 2^{\log_2(r) - 1 + 1} = 2^{\log_2(r)} = r.$$

But $T$ was defined to have $r$ leaves. This contradicts the assumption. □

The *least common ancestor* of a pair of nodes $v_i$ and $v_j$, is the node where the paths from $v_i$ to the root and from $v_j$ to the root both meet. If $v_i$ and $v_j$ are leaves then it is the node closest to $v_i$ and $v_j$, from which both $v_i$ and $v_j$ are descended. A *subtree* of a tree $T$ is a subgraph of $T$ that is also a tree. The only property of a tree that does not automatically hold for any subgraph is connectivity, so a subgraph of a tree is only a subtree if it is connected. A *forest of trees* is a collection of two or more trees with non-intersecting node

sets. For example, if we remove the root, and all edges connected to it, from a complete binary tree, then we are left with a forest of two trees. Not only are both of these trees subtrees of the original complete binary tree, they are also complete binary trees in their own right. Since all the leaves were the same distance to the root in the first tree, they are the same distance to their respective roots in the forest. The internal degrees do not change, except for the new roots, both of which have degree two. Indeed, any node in a complete binary tree can be chosen to be the root of a complete binary subtree.

Some of these properties are shown in Figure 2.1. As is customary, nodes at the same depth from the root are all drawn horizontally. A *level* in a tree is the collection of all nodes that are the same distance from the root. The binary tree on the left is not a complete tree, but the subtree rooted at the right child of node $v$ is. The complete tree is labelled according to *breadth first* labelling. The root is labelled 1, and we increase the label by one for each node on the level beneath (going left to right) until we reach the last leaf. The advantage of this labelling scheme is that the children of node $v$ are $2v$ and $2v+1$. Using this notation, if $v$ is at height $h$ then the descendants of a node $v$ are simply $\{2^h v, \ldots, 2^h v + 2^h - 1\}$. We will sometimes refer to this set as $desc(v)$. Of course, this only works for a complete binary tree and breadth first labelling.

### 2.3.2 Steiner Trees

There is another type of graph that we will use frequently. A *Steiner Tree* is a subgraph of a tree, and is defined by a set of nodes from that tree. To generate the Steiner Tree, we take the set of nodes $\mathcal{R}$, and find all the paths from each of these nodes to the root. The node set of the Steiner Tree (or $ST(\mathcal{R})$) is the set of all nodes that occur in these paths, and similarly the edge set is the set of all edges that occur. Since it is connected and is a subgraph of a tree, $ST(\mathcal{R})$ is also a tree. Since the path between any two nodes in a tree is unique, the tree $ST(\mathcal{R})$ is uniquely defined by $\mathcal{R}$. In Figure 2.1, the tree on the right is the Steiner Tree of nodes $a$, $b$ and $c$. There is a very simple result

for Steiner Trees based on binary trees:

*Lemma* 10. Let $\mathcal{R}$ be a non-empty set of leaves in a complete binary tree. Then the number of nodes in $ST(\mathcal{R})$ with both children in $ST(\mathcal{R})$ is $|\mathcal{R}| - 1$.

*Proof.* Consider the number of nodes in $ST(\mathcal{R})$ at each level, starting at the root. By the definition, the root is in $ST(\mathcal{R})$. At the second level, the number of nodes in $ST(\mathcal{R})$ is one if the root has one child in $ST(\mathcal{R})$, but is two if both children are. If, in a given level of $ST(\mathcal{R})$, there are $a$ nodes, $b$ of which have two children in $ST(\mathcal{R})$, then the number of nodes in $ST(\mathcal{R})$ in the next level of $ST(\mathcal{R})$ is $a+b$. Let $a_i$ be the number of nodes of $ST(\mathcal{R})$ at level $i$ and let $b_i$ be the number of nodes in $ST(\mathcal{R})$ at level $i$ that have two children in $ST(\mathcal{R})$ at level $i+1$. Then $a_{i+1} = a_i + b_i$ for $i = 0, \ldots, k-1$ and so

$$a_k = a_0 + \sum_{i=0}^{k-1} b_i.$$

Since the root is the only node in $ST(\mathcal{R})$ at the level $i = 0$, $a_0 = 1$. And by the definition of $ST(\mathcal{R})$, the number of nodes at level $i = k$ is $|\mathcal{R}|$, i.e. $a_k = |\mathcal{R}|$. The sum of the $b_i$'s is the total number of nodes in $ST(\mathcal{R})$ that have both children in $ST(\mathcal{R})$, which is what we are looking for. Therefore:

$$\text{Number of nodes } v \text{ s.t. both children } \in ST(\mathcal{R}) = |\mathcal{R}| - 1. \qquad \square$$

As $ST(\mathcal{R})$ is a subtree of the complete tree $T$, there will be some nodes in $T$ that are not in $ST(\mathcal{R})$ (assuming $\mathcal{R}$ is a proper subset). We define a node $v_i$, to be *hanging off* (or *hangs off*) if $v_i$ is not contained in $ST(\mathcal{R})$ but $par(v_i)$ is in $ST(\mathcal{R})$. This is an important property of the subtree that will be used in forming the cover in several Revocation Schemes.

One result that we will need later concerns the least common ancestor for Steiner Trees. For any set of leaf nodes $\mathcal{R}$ on a binary tree, we can always find at least one pair of nodes in $\mathcal{R}$ such that their least common ancestor has no other descendants in $\mathcal{R}$. Consider the Steiner Tree in Figure 2.1 ($\mathcal{R} = \{a, b, c\}$). The diagram shows the least common $b$ and $c$, but this node is also the ancestor of $a$. The least common ancestor of $a$ and $b$ is not an ancestor of

**Algorithm to find the least common ancestor of only two nodes**

```
0:  Initialise: T, a binary tree, node v = root, R, subset of at
      least two leaf nodes of T
1:  while v has more than two nodes from R as descendants do
2:    if v only has one child in ST(R) then
        set v to be this child
3:    else
4:      if one child of v has exactly one descendant in R then
          set v to be this child's sibling
5:      else
6:        set v to be the child that has the least descendants in R
            (or either child of v if both are the same)
7:      end if
8:    end if
9:  end do
10: while v is not the least common ancestor of the (two) nodes
      in R that are descended from it do
11:   set v to be its child that is in ST(R)
12: end do
```

Table 2.1: Algorithm to find a node $v$, the least common ancestor of two leaves in $\mathcal{R}$ such that $v$ has no other descendants in $ST(\mathcal{R})$.

the only other node in $\mathcal{R}$, $c$, and so is of the form we want. Table 2.1 gives an algorithm to find such a node. The nature of the stopping conditions in the algorithm ensure that the output will be in the form we want, a node that is the ancestor of exactly two nodes of $\mathcal{R}$ and is their least common ancestor. The proof that the algorithm works is mainly concerned with showing that the algorithm does actually terminate.

*Lemma* 11. Let $T$ be a (finite) binary tree and $\mathcal{R}$ a subset of nodes of $T$ (size at least 2). Then we can always find at least one pair of nodes in $\mathcal{R}$ such that their least common ancestor has no other descendants in $\mathcal{R}$.

*Proof.* Let $ST(\mathcal{R})$ be the Steiner Tree connecting the nodes in $\mathcal{R}$ and the root. Apply the above algorithm on the set of nodes $\mathcal{R}$. The output of the algorithm $v$, has to have less than or equal to two nodes in $\mathcal{R}$ descended from it (by the conditional statement of the while loop). However, the algorithm

never chooses a node with only one descendent in $\mathcal{R}$. Therefore, if the while loop terminates, $v$ will have exactly two descendants in $\mathcal{R}$. It just remains to show that the loop will terminate.

Firstly, we will show that every iteration through either *while* loop, moves $v$ to a node further from the root. We know that there are at least 3 nodes from $\mathcal{R}$ descended from $v$ at the start of any iteration of the first loop. We will call the set of nodes from $\mathcal{R}$ that are descended from $v$, $\mathcal{R}'$. Of the two children of $v$, one of the following must be true:

1. All nodes in $\mathcal{R}'$ are descended from one of the two children.

2. One child node has exactly one node from $\mathcal{R}'$ as a descendant, the other having the rest (at least two).

3. Both children have more than one node from $\mathcal{R}'$ as a descendant.

In each case, we have a corresponding action in the algorithm that takes us to one of the two child nodes. For the first case we have Step 2, the second we have Step 4, and for the last case we have Step 6. So we are moving further from the root in each step. The number of nodes from $\mathcal{R}$ descended from $v$ is bounded above by the total number of nodes descended from $v$. This eventually gets reduced to 2 when we get to the parents of the leaves, so the first while loop must terminate at some node by then. Since the second while loop also moves $v$ to a node further from the root every iteration, it also terminates. □

We have now introduced the terminology and notation we will be using in our analysis of Revocation Schemes. Of particular importance will be Formulae (2.2) and (2.3) for $t_{\max}(n, r)$ and $t_{aver}(n, r)$. The two properties of Revocation Schemes that we will be examining will be bandwidth, in terms of these two measures, and storage, in terms of $|U|_{\max}$.

# Chapter 3

# Previous Work

In this chapter we will summarise the existing work in the field of Broadcast Encryption, as well as some related topics. Those subjects that we will be looking at in later chapters will be described in detail in subsequent sections.

One such related topic is that of *Key Predistribution Systems*. The problem can be considered a special case of that addressed by Revocation Schemes. The centre has to distribute secret information to each user in such a way as to allow any subset of users to establish a common key. The Revocation Scheme of Lemma 7 is essentially a Key Predistribution System as for every subset of users there is one secret key known only to them and the centre. The centre does not play any further part in a Key Predistribution Systems after initialisation, as the users play a more active role. There may need to be interaction between the users to establish the key, and it is the users who communicate the messages, i.e. it allows many-to-many multicast, rather than just one-to-many broadcast. The first work on this subject was by Blom [7], who designed a system using MDS codes that allowed pairs of users to generate common keys. Matsumoto and Imai [20] constructed the first general system using symmetric matrices. A more complete review of the topic can be found in [17], and recent work on the subject include [25] and [4].

One of the first papers to address the problem of a centre who wants to broadcast to a group was by Berkovits [5]. He proposed adapting a Secret Sharing Scheme to solve the problem. Secret Sharing Schemes were discovered independently by Blakley [6] and Shamir [28], and solve a problem slightly

different to that in Broadcast Encryption (but with some similarities). A centre (usually called a dealer in Secret Sharing Schemes) wants to give users "shares" of a secret in such a way that only certain subsets of users can combine their shares to recover the secret. The most common is an $(m, n)$-threshold Secret Sharing Scheme where any $m$ of the $n$ shares are required to recover the secret. Berkovits' solution removed the need for users to combine their shares, but as the secret is an integral part of the Secret Sharing Scheme, it could only be used once. The shares must be updated after every use.

The phrase Broadcast Encryption was first coined by Fiat and Naor [13]. They defined it to mean any situation where a centre distributes keys to users to allow broadcasting to subsets of users. They also defined *$k$-Resilient Schemes*, which are a more relaxed version of Revocation Schemes. In a $k$-Resilient Scheme, the goal of the centre is the same as in a Revocation Scheme, namely distribute content to only those users in a privileged subset. The difference is that a $k$-Resilient Scheme has a more relaxed security condition. The centre constructs the broadcast in such a way that no coalition of $k$ revoked users can collude to decrypt the message. They describe several 1-Resilient Schemes (one based on one-way functions, one based on extracting roots modulo composites). They then explain how to construct $k$-Resilient Schemes by combining 1-Resilient Schemes using perfect hash functions. Their best scheme required storage of $\mathcal{O}(k \log k \log n)$ and the centre to broadcast $\mathcal{O}(k^2 \log^2 k \log n)$ messages. However, these complexities are too high for useful applications as commercial pirates could have access to large numbers of legitimate receivers (e.g. decoder boxes, smart cards, etc.). Further work include [15], [29] and [11]. The Key Predistribution Systems of [25] are generalised to Broadcast Encryption schemes in [26].

About the same time Chor, Fiat and Naor introduced the concept of Traitor Tracing Schemes [9]. In this scenario the centre wishes to broadcast a secret so that all users can decode it, but enabling the centre to find the source of any leak should the secret be disclosed. The secret is split into shares, each of which is encrypted with a number of secret keys. For each encrypted share, all users will have exactly one secret key to decrypt it. However, the key they have will differ from user to user. It is these keys that the users use to

decrypt the shares that will identify the user if they are disclosed, even if a number of users collude to form a "pirate decoder". In one of the schemes they propose, an $(l, k^2)$ table of secret keys is constructed. The secret is divided into $l$ shares (i.e. $l$ values that XOR to give the secret), and the $i^{th}$ share is encrypted with each key in the $i^{th}$ row of the table. Each user is given one key per row, assigned by random hash values. Since the personal keys of the users are different from those selected by other users in the vast majority of the rows, it is still possible to identify the traitors even when keys are pooled. The tracing algorithm can be found in [9] and further work in [24], [22] and [14].

The Logical Key Hierarchy (LKH) scheme devised by Wallner et al. [10], and independently by Wone et al. [31], is a Broadcast Encryption scheme for Stateful Receivers. The object of the scheme is mostly the same as that of a Revocation Scheme, but the implementation is different. The users are not passive receivers to be deemed privileged or excluded by the centre, but rather active participants who can request to leave or join at will. To initialise the scheme, the centre will assign the group of users to the leaves of an $a$-ary tree. The centre will allocate keys to each node in the tree, and each user receives the keys for all nodes on the path from its leaf to the root. At certain times, the centre will broadcast an encrypted message. Unlike a Revocation Scheme, only one of the keys in the tree is used in the broadcast, that of the root. The purpose of the other keys is to allow the centre to update the keys in the event of a users leaving or joining the group. If a user is to join the group, a new leaf has to be added to the tree. All keys for nodes on the path from the new leaf to the root have to be changed to prevent the new user accessing previous broadcasts. Each of these keys is encrypted and broadcast, both with the old key for the existing users, and with the new user's key, making $2\log_a(n)$ encryptions in total. When a user leaves the group, all keys that user had must be updated. Obviously these keys cannot be used to encrypt the new keys. Instead the $a$ keys one level down are used. So to remove a user from the group means sending $(a-1)\log_a(n)$ encrypted messages.

The obvious benefit of such a scheme over a Revocation Scheme is the ability to broadcast the encrypted message without a header. The cost is the

effort, for the users as well as the centre, needed to add or remove anyone from the group. The workload of the centre can be reduced by updating the group several users at a time, or "batch re-keying", as studied in [18].

Luby and Staddon looked at a variety of Revocation Protocols in [19], and provided various combinatorial bounds. The bounds only apply to information theoretically secure schemes. As most of our schemes have security based on computational assumptions, the bounds do not apply. They also defined two protocols, the "OR Protocol", which is what we will be focusing on, and the "AND Protocol". Both methods use a broadcast key to encrypt the message, but encrypt the broadcast key in different ways. In the AND Protocol, it is necessary for a user to have all the keys that were used to encrypt the broadcast key in order to decrypt the message. With the OR Protocol a user only needs to have one of them. The major difference between the two is that the OR Protocol is resilient against any coalition of excluded users, while the AND Protocol is not.

The Revocation Protocol defined in Chapter 2 is an instance of the "OR-Protocol". Any user need only know one of the keys $L_{i_j}$ used in the broadcast to get the session key and hence the message. It is therefore resilient against arbitrary coalitions of revoked users, since no revoked user has any of the keys used. However, this says nothing about privileged users conspiring with revoked users. If any privileged user redistributes the message, or even shares his private keys with other users, then some users in $\mathcal{R}$ may be able to gain access to the message. This is what Traitor Tracing schemes are designed to prevent.

Naor, Naor and Lotspiech described two Revocation Schemes in [23] that will be the basis for a lot of the work we will be doing. The first, the Complete Subtree Revocation Scheme, has similarities to the Logical Key Hierarchy. Users are assigned to the leaves of (binary) tree, and keys are assigned to each node of the tree. Since the receivers are stateless, the keys do not get updated as happens in LKH. When the centre wants to broadcast to a subset of users it can choose keys from the tree that only belong to users in the subset. The details of their scheme are more thoroughly described in Section 3.1, and we further develop these ideas in the Chapter 4. Asano looked at the Complete

Subtree Revocation Scheme based on an $a$-ary tree [1], and we will describe his work in more detail in Section 3.2.

The second scheme of Naor et al. was the Subset Difference Revocation Scheme. This was based on a binary tree, much in the same way as the Complete Subtree Revocation Scheme. The important difference was that keys were assigned to pairs of nodes on the tree. We give the details of this scheme in Section 3.3. The methods of Naor et al. were extended to the public-key environment by Dodis and Fazio in [12], by the use of Hierarchical Identity-Based Encryption. This would be necessary if the Key-Generation centre wanted other (untrusted) parties to be able to broadcast.

The Subset Difference Revocation Scheme was generalised in the paper by Halevy and Shamir [16], to give the Layered Subset Difference (LSD) scheme. Only a subset of the keys in the Subset Difference Revocation Scheme are used in LSD, dictated by special "layers" in the tree. By the nature of the layers, any key that would have been used in the Subset Difference Revocation Scheme can be replaced by either one or two keys in LSD. This significantly reduces the storage requirement at the cost of doubling the bandwidth. Two papers by Asano present methods to slightly reduce the storage requirement without sacrificing the bandwidth are [2] and [3], as well as [4]. An interesting way of combining schemes in presented in [21]. The presented scheme (combination of Complete Subtree and LSD) appears to form a good balance between parameters of the component schemes.

A modification of the Subset Difference Revocation Scheme is given in [30]. While not strictly for stateless receivers, it does improve on the original. The basic idea is to assign users to several miniature versions of the Subset Difference as they join, and permanently revoking leaves of departed users. As soon as one scheme is entirely comprised of departed users, it can be removed from the system. This is speeded-up by "shifting" users out of sparsely populated schemes, by unicasting replacement keys to the users. This requires the receivers to be stateful, but the smaller schemes mean less storage, and the flexibility reduces communication costs. Another variation is [32], which adds the properties of self-healing ([29]) and reliability to the Subset Difference Revocation Scheme.

In the paper by Chen and Dondeti [8], both the stateful scheme, LKH, and the stateless scheme, the Subset Difference Revocation Scheme, are simulated under different circumstances. The results show that frequent, minor changes to the system (users entering and leaving the system) favour the stateful scheme, but with less regular and more dramatic changes the stateless scheme performs better.

## 3.1 Complete Subtree Revocation Scheme

We now come to the first of the two schemes put forward by Naor, Naor and Lotspiech [23]. In this section, we will briefly describe the scheme, as well as the relevant details from [23]. These include the process for forming a cover of $\mathcal{N} \setminus \mathcal{R}$ and a bound on $t_{\max}(n, r)$.

The Complete Subtree Revocation Scheme is the first of many *tree-based* Revocation Schemes that we will be looking at. What we mean by "tree-based" is that each user is assigned a leaf on some rooted-tree, and that the set $\Omega$, and the functions $\gamma$ and $\delta$, are defined by the nodes and edges of the tree. While the LKH scheme was also tree-based, it was not a Revocation Scheme as we have defined it, since it relied on Stateful Receivers.

To initialise the system, the centre must assign an index to each node (including leaves) in the tree. The Complete Subtree Revocation Scheme for $n = 2^k$ users is based on the complete binary tree with $2^k$ leaves. The index set is $\Omega = \{1, 2, \ldots, 2^{k+1} - 1\}$, and so the corresponding set of establishment keys is $\mathcal{L} = \{L_1, L_2, \ldots, L_{2^{k+1}-1}\}$. The set of users who share any key is simply all the users who are assigned a leaf that is descended from the node for that key's index. If the node is $v_i$, then this corresponds to all the leaves of the subtree rooted at $v_i$. So for any key $L_i$, the function $\gamma(i)$ first identifies the node on the tree that index is assigned to, and then returns the users corresponding to all leaves descended from that node. We formally define the scheme as follows:

*Definition* 12. A Complete Subtree Revocation Scheme $(\mathcal{N}, \Omega, \gamma_f)$ on $n = 2^k$ users is defined as follows. Let $T$ be a complete binary tree with $2n - 1$ nodes $\{v_1(= root), \ldots, v_{2n-1}\}$, indexed using breadth first labelling. Let $desc(v_i)$ for $i \in \{1, \ldots, 2n - 1\}$ denote the subset of leaves that are descendants of the

node $v_i$.

$$\begin{aligned} \mathcal{N} &= \{u_1, \ldots, u_n\} \\ \Omega &= \{1, \ldots, 2n-1\} \end{aligned}$$

Let $f$ be a bijection that maps leaves to users:

$$f : [v_n, \ldots, v_{2n-1}] \to \mathcal{N}.$$

We define $\gamma_f$ of any index $i$ as follows:

$$\gamma_f(i) = \{f(v_l) : l = 2^h i, \ldots, 2^h i + 2^h - 1\}, \text{ where } h \text{ is the height of node } v_i,$$

or equivalently $\gamma_f(i) = \{f(v_l) : v_l \in desc(v_i)\}$.

Note: (1) We are assuming the number of users is a power of 2.

(2) The leaves of a complete binary tree with breadth first labelling are $v_n, v_{n+1}, \ldots, v_{2n-1}$. The set of users is $\{u_1, u_2, \ldots, u_n\}$. So the simplest example of $f$ is just:

$$f(v_i) = u_{i-(n-1)}.$$

We will assume $f$ is defined as above unless stated otherwise.

Each user is given all keys on the path from their leaf to the root, so:

$$|U|_{max} = \log_2(n) + 1. \tag{3.1}$$

Because we are dealing with a complete binary tree, the number of users that share the key for any given node will always be a power of 2. Specifically, if that node is at a height of $h$ from the leaves then $2^h$ users will share the key for that node. There are two specific key types of note:

- The key for the root, which is shared by all users
- The key for any leaf, which is only known to one user (each user has one such key)

The former key ensures that whenever the centre wishes to broadcast to the entire user set ($r = 0$), it will only need to use one establishment key ($t_{\max}(n, 0) = 1$). The latter keys satisfy the requirement in Theorem 4:

$$\gamma_f(i) = \{f(v_i)\} \text{ for } i = n, \ldots, 2n-1.$$

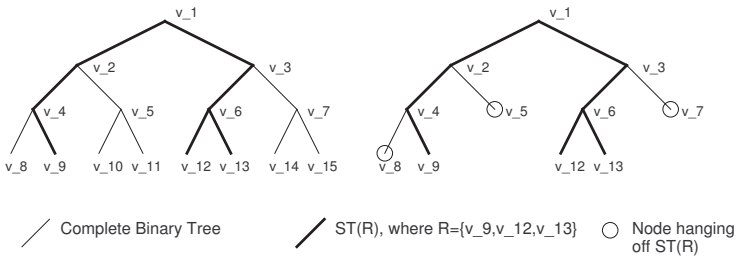


Figure 3.1: A complete binary tree and $ST(\{9, 12, 13\})$

So for any choice of revoked users, at the very least we have the ability to use one establishment key for each remaining privileged user ($t_{\max}(n, r) \leq n - r$). We shall now see how to revoke any subset of users of any size using significantly fewer subsets in the cover.

Whenever the centre has a message to send and a subset of users, $\mathcal{R}$, who are to be excluded, it creates a Steiner Tree that connects all excluded users and the root, $ST(\mathcal{R})$. The keys used in the broadcast are precisely the ones associated with nodes that just "hang off" $ST(\mathcal{R})$. As we defined in Section 2.3.2, by "hanging off" we mean any node that is not in $ST(\mathcal{R})$, but whose parent is. These nodes can be seen to cover $\mathcal{N} \setminus \mathcal{R}$ (see Lemma 13). An example of this is given in Figure 3.1. In this case the keys that would be used in the broadcast are $L_5, L_7, L_8$.

In a Complete Subtree Revocation Scheme (as well as later schemes), $\mathcal{R}$ is a subset of users, not leaves. But since we will be working so closely with the properties of binary trees, for the rest of this chapter we will refer to leaves directly, rather than "leaves corresponding to users". Consequently, leaves that correspond to privileged/revoked users will be referred to as privileged leaves/revoked leaves.

Another way of describing the above cover is to consider the result of deleting all nodes and edges of $ST(\mathcal{R})$ from the original tree. This will give a forest (collection of trees), the leaves of which correspond to all the privileged users (and only privileged users). Applying $\gamma$ to the index of the root of each of these trees gives us the cover of $\mathcal{N} \setminus \mathcal{R}$. The following result was stated in [23], and we give the complete proof.

*Lemma* 13. Let $(\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme with

$n = 2^k$ users, determined by the correspondence between the users and the leaves of a complete binary tree $T$. For any $\mathcal{R}$, let $C$ be the set of indices of all nodes that hang off $ST(\mathcal{R})$. Then $\gamma(C)$ forms a cover:

$$\bigcup_{i \in C} \gamma(i) = \mathcal{N} \setminus \mathcal{R},$$

and $t(\mathcal{N}, \mathcal{R}) = |C|$.

*Proof.* In order to prove that the union of subsets is a cover, we need to show that all privileged users are contained in the union, and no revoked users are. We can define the set of indices used in the cover to be:

$$\{i : v_i \notin ST(\mathcal{R}), par(v_i) \in ST(\mathcal{R})\}.$$

Consider any privileged user $u \in \mathcal{N} \setminus \mathcal{R}$. Let $v_l$ be the leaf in tree $T$ associated with that user. Since the only leaves in $ST(\mathcal{R})$ correspond to revoked users, $v_l$ is not in $ST(\mathcal{R})$. As the root is contained in $ST(\mathcal{R})$, on the path from $v_l$ to the root, there must be some node $v_i$ such that $v_i \notin ST(\mathcal{R})$ and $par(v_i) \in ST(\mathcal{R})$. Therefore $i \in C$. Since $\gamma(i)$ is just the set of users who's leaves are descended from $v_i$, $u \in \gamma(i)$ for some index $i$ in $C$. This applies to all privileged users. Any revoked user's leaf belongs to $ST(\mathcal{R})$. The path from this leaf to the root is unique, and $ST(\mathcal{R})$ is comprised of all paths from revoked users' leaves to the root, so all nodes on a path from such a leaf to the root are in $ST(\mathcal{R})$. Therefore no revoked user is in the cover (none of the nodes on the path from a revoked leaf to the root belong to $C$). Thus:

$$\bigcup_{i \in C} \gamma(i) = \mathcal{N} \setminus \mathcal{R}.$$

In order to show that $t(\mathcal{N}, \mathcal{R}) = |C|$, we must also show that the cover generated is also minimal. Consider any other cover of $\mathcal{N} \setminus \mathcal{R}$, $C'$. If $C' \subset C$, let $k$ be some index in $C$ but $k \notin C'$. By the definition of $C$, there is some leaf, $v_l$, descended from $v_k$ and associated with a privileged user. We also have that all ancestor nodes of $v_k \in ST(\mathcal{R})$, as $ST(\mathcal{R})$ is a collection of paths to the root. So no ancestor of $v_k$ can hang off $ST(\mathcal{R})$. And since no descendants of $v_k$ are in $ST(\mathcal{R})$, no descendant of $v_k$ can hang off $ST(\mathcal{R})$ either. Therefore, $v_k$ is the only node that hangs off $ST(\mathcal{R})$ on the path from $v_l$ to the root.

Since $v_k \notin C'$, there is at least one leaf corresponding to a privileged user that is not descended from any node in $C'$. Therefore, $C'$ is not a cover.

Alternatively, if $C' \not\subset C$, let $j$ be some index in $C'$, but $j \notin C$. There are two possibilities for $j$, either $v_j \in ST(\mathcal{R})$ or $v_j$ is descended from $v_i$, where $i \in C$. Since $ST(\mathcal{R})$ is a collection of paths from leaves in $\mathcal{R}$ to the root, if $v_j \in ST(\mathcal{R})$ then there is some leaf in $\mathcal{R}$ descended from $v_j$. This will lead to a revoked user in the cover, which is expressly prohibited. The remainder of the tree $T$ is a forest of complete rooted subtrees. For the root of any such subtree $v_i$, $i \in C$. So if $j$ is not in $ST(\mathcal{R})$ and $j \notin C$, then $v_j$ must be descended from some $v_i$. Since there are privileged leaves descended from $v_i$ not descended from $v_j$, in order for $C'$ to be a cover, there must be at least one more index in $C'$ for the portion of $T$ descended from $v_i$ than there is in $C$. Therefore, $|C'| > |C|$, and so $\gamma(C)$ is a minimal cover. □

Any node on $ST(\mathcal{R})$ that has degree 3 (counting edges on $ST(\mathcal{R})$ only, not on the original complete tree), one path from above and forking in two below, does not have any nodes hanging off. An internal node on $ST(\mathcal{R})$, with degree 2, only has one child that is in $ST(\mathcal{R})$ and hence leads to revoked user(s). The other child is not in $ST(\mathcal{R})$ and can only lead to privileged users and it is the first node that we encounter on this path that we say "hangs off" $ST(\mathcal{R})$. A node on $ST(\mathcal{R})$ with degree 1 must be a leaf, as all paths on $ST(\mathcal{R})$ only terminate at leaves. If it is a leaf then it has no descendants, which means no nodes "hanging off". The root is an exception to these rules, since it is the only node in $ST(\mathcal{R})$ without a path entering it from above. So the root has one node hanging off if it has degree 1 and no nodes hanging off if it has degree 2. The root will not have degree 0 for a non-trivial $ST(\mathcal{R})$. A more succinct way of saying the above is that a node in $ST(\mathcal{R})$ will have a node hanging off if and only if its degree in $ST(\mathcal{R})$ is strictly less than its degree in the original tree. If the node is in $ST(\mathcal{R})$, then there will be a path from the root to this node in both trees. The difference in degrees means that (at least) one of the edges to the child nodes will not be present in $ST(\mathcal{R})$, and this missing edge gives us a node hanging off. In later chapters, we will be using the difference in degrees between the two trees when talking about nodes in $ST(\mathcal{R})$ that have nodes hanging off. Unlike the degree of the node in $ST(\mathcal{R})$,

we do not have to make different statement for the root (as it has degree one less).

In [23], Naor et al. used an inductive argument to prove a bound on $t_{\max}(n, r)$:

$$t_{\max}(n, r) \leq r \log_2(n/r). \tag{3.2}$$

In their paper they give a sketch proof. The complete proof can be found in Appendix A.

It is possible to construct examples that give equality in the bound of Formula (3.2). For $r$ set to a power of 2 less than $n$, we can construct a set $\mathcal{R}$ of size $r$ such that $t(\mathcal{N}, \mathcal{R}) = r \log_2(n/r)$ (we shall see the reason for this in Chapter 4). However, it is also simple to see that, for some values of $r$, $t_{\max}(n, r)$ is strictly less than $r \log_2(n/r)$. This follows from the result Lemma 5, that

$$t_{\max}(n, r) \leq n - r < r \log_2(n/r) \quad \text{for } n/2 < r < n.$$

One of the first things we will do in Chapter 4 is to derive a formula for $t_{\max}(n, r)$, as well as investigate the properties of $t_{aver}(n, r)$. We will also look into different ways of improving the scheme, by generalising it to an $a$-ary tree, to a forest of trees, and deriving more efficient methods of storing keys.

## 3.2 Complete Subtree on an $a$-ary tree

In this section, we consider Asano's generalisation of the Complete Subtree to an $a$-ary tree [1]. This variation leads to a much greater storage at the receiver's end. As well as giving the definition and some of the bounds on $t_{\max}(n, r)$, we will also describe the two methods proposed by Asano to reduce the storage.

The obvious way to generalise the Complete Subtree Revocation Scheme would be to assign the users to the leaves of the $a$-ary, have one key for each node on the tree and each user possesses the keys on the path from their leaf to the root. Storage is reduced as each user only needs $\log_a(n) = \frac{\log_2(n)}{\log_2(a)}$ keys, compared to $\log_2(n)$ for the Complete Subtree. However, since each user belongs to fewer subsets, more subsets will be required to form the covers. The

upper bound of $t_{\max}(n,r) \leq n/2$ does not hold anymore, instead we can only guarantee that $t_{\max}(n,r) \leq (a-1)\frac{n}{a}$. The most keys needed for any cover out of a group of $a$ leaves descended from the same node is $a-1$, and there are $\frac{n}{a}$ such nodes. Asano proposed a scheme that traded-off storage and bandwidth in the opposite direction by combining the $a$-ary tree with the idea of a "Power Set Method".

What Asano refers to as the Power Set Method is the scheme described in Lemma 7. This scheme had the absolute minimum bandwidth as there was a key for every possible revoked set of users. The flaw was of course the exponential storage requirement. Asano proposed applying this idea to each node in the $a$-ary tree and its children, rather to the whole set of $n$ users. All internal nodes in the tree have $a$ children. A total of $2^a - 2$ subsets are defined for each node, all possible non-empty proper subsets of children. Each subset of children of each node will have its own key, and this key will be given to any user who is descended one of the children in the subset. Since we are dealing with an $a$-ary tree, we can no longer identify children of a node by left and right. Instead we refer to them in order from left to right as $1^{st}$ child, $2^{nd}$ child, up to $a^{th}$ child. We formally define the scheme as follows:

*Definition* 14. A Complete Subtree Revocation Scheme, $(\mathcal{N}, \Omega, \gamma_f)$, on an $a$-ary tree with $n = a^k$ users is defined as follows. Let $T$ be a complete $a$-ary tree with $\frac{an-1}{a-1}$ nodes, $\{v_1(= root), \ldots, v_{\frac{an-1}{a-1}}\}$, indexed using breadth first labelling. Let $desc(v_i)$ for $i \in \{1, \ldots, \frac{an-1}{a-1}\}$ denote the subset of leaves which are descended from the node $v_i$.

$$\begin{aligned} \mathcal{N} &= \{u_1, \ldots, u_n\} \\ \Omega &= \left\{1, \ldots, \frac{n-1}{a-1}\right\} \times \{b_1 b_2 \ldots b_a : b_i \in \{0,1\}, \sum_{i=1}^{a} b_i \neq 0 \text{ or } a\} \end{aligned}$$

Let $f$ be a bijection that maps leaves to users:

$$f : [v_{\frac{n-1}{a-1}}, \ldots, v_{\frac{an-1}{a-1}}] \rightarrow \mathcal{N}.$$

We define $\gamma_f$ of any index $(i, b_1 b_2 \ldots b_a)$, in terms of the children of the node $v_i$ :

$$\gamma_f(i, b_1 b_2 \ldots b_a) = \{f(v_l) : v_l \in desc(j^{th} \text{ child of } v_i), \text{ and } b_j = 1\}.$$

We also add the following index to $\Omega$:

$$\gamma_f(root, 11\ldots1) = \mathcal{N}.$$

We leaves out the bits $b_1b_2\ldots b_a$, that sum to 0 as that would correspond to a key that no user has. The reason we leave out the bits that sum to $a$ is because it is redundant. A subset containing all leaves descended from all the children of a node $v_i$ is the same as the subset containing all the leaves descended from one of the children of the parent of $v_i$, which is already in the scheme:

$$\gamma_f(i, 111\ldots1) = \gamma_f(par(v_i), B),$$

where $B$ is 1 corresponding to the relationship between $par(v_i)$ and $v_i$, and zero elsewhere. This works for all nodes, except for the root (has no parent), which is why we add the last index.

The difference applying the Power Set Method makes is that no matter how many privileged children a node in $ST(\mathcal{R})$ has, there need only be one index to cover all nodes hanging off. With the simpler $a$-ary tree scheme any one node in $ST(\mathcal{R})$ could require up to $a-1$ indices for its privileged children (would need $a-1$ if there is only one child of the node revoked). Consequently, each node directly above the leaves can only require 1 index to cover all its privileged children. As there are only $n/a$ such nodes, we have:

$$t_{\max}(n, r) \leq \frac{n}{a}. \tag{3.3}$$

This is a clear improvement over the Complete Subtree Revocation Scheme for any $a > 2$. Asano also showed that:

$$t_{\max}(n, r) \leq r\left(\frac{\log(n/r)}{\log(a)} + 1\right).$$

We will derive the exact value of $t_{\max}(n, r)$ in Chapter 4.

The down-side of this scheme is the cost of storage. The number of nodes on the path from any leaf to the root is $\log_a(n) + 1 = \frac{\log_2(n)}{\log_2(a)} + 1$. Each user must store $2^{a-1} - 1$ keys for each node on the path from their leaf to the root. On top of this, there is one extra key for the root for all users. That gives:

$$|U|_{\max} = 1 + (2^{a-1} - 1)(\log_a(n)) = 1 + (2^{a-1} - 1)\frac{\log_2(n)}{\log_2(a)}. \tag{3.4}$$

The fact that this is exponential in $a$ is undesirable. In order to make the amount of data each user must store more manageable, Asano proposed two Methods of key generation. Both techniques are based on the RSA cryptosystem [27], and work over a suitable public modulus $M = (q_1, q_2)$.

### 3.2.1 Compression Method 1

The Broadcast centre chooses $(2^a - 2)\frac{n-1}{a-1} + 1$ primes, $p_{i,b_1b_2\ldots b_a}$, one for every index $(i, b_1b_2\ldots b_a) \in \Omega$. Let $B$ denote $b_1b_2\ldots b_a$. The centre must publish both the list of primes and the assignment of $p_{i,B}$ to $(i, B) \in \Omega$. It then chooses a random number $K \mod M$, and sets the establishment key $L_{i,B}$ to be:

$$L_{i,B} = K^{T/p_{i,B}} \mod M,$$

where $T = \prod p_{i,B}$ is the product of all the primes. Each user $u_j$ is then given a unique Master Key $MK_j$ from which any of their $(2^{a-1} - 1)\log_a(n) + 1$ establishment keys can be generated. Let $w_j$ be the product of all primes of the form $p_{i,B}$, where $u_j \in \gamma(i, B)$. The Master Key $MK_j$ for user $u_j$ is defined as:

$$MK_j = K^{T/w_j} \mod M.$$

To generate any key $L_{i,B}$ from $MK_j$, $u_j$ calculates the product of all primes assigned to the subsets he/she belongs to, with the exception of $p_{i,B}$ (the primes are public). Since this is $w_j/p_{i,B}$, they can work out:

$$MK_j^{w_j/p_{i,B}} \mod M = \left(K^{T/w_j}\right)^{w_j/p_{i,B}} \mod M = K^{T/p_{i,B}} \mod M = L_{i,B}.$$

Asano proved that this method is secure against any conspiracy of revoked users under the assumption that computing $p^{th}$ roots ( $\mod M$) is difficult.

This method requires $(2^{a-1} - 1)\log_a(n)$ multiplications and one modular exponentiation. The need to have access to the primes is an added cost. If the users only retrieve them from the centre as needed, we create another bottleneck on the bandwidth. The user could store only those primes they will use. As the total number of primes is $(2^a-2)\frac{n-1}{a-1}+1$, the size of the primes is roughly $\mathcal{O}(2^a n \log(2^a n))$. Each user needs only $(2^{a-1} - 1)\log_a(n) + 1$ primes, which means the storing of $\mathcal{O}\left((\frac{2^{a-1}-1}{\log(a)}\log(n) + 1)(\log(n) + a + \log(\log(n) + a))\right)$ bits.

| | $CSRS$ | $SDRS$ | Method 1 | Method 2 |
|---|---|---|---|---|
| $t_{\max}(n,r)$[1] | $r\log(n/r)$ | $2r-1$ | $r\left(\frac{\log(n/r)}{\log(a)}+1\right)$ | $r\left(\frac{\log(n/r)}{\log(a)}+1\right)$ |
| $\lvert U\rvert_{\max}$ | $\log(n)$ | $\mathcal{O}(\log^2(n))$ | 1 | $\frac{\log(n)}{\log(a)}$ |
| PRNG | - | $\mathcal{O}(\log(n))$ | - | - |
| Gen. of primes | - | - | $\mathcal{O}\left(\frac{2^a\log^5(n)}{\log(a)}\right)$ | - |
| No. of multi. | - | - | $\frac{(2^{a-1}-1)\log(n)}{\log(a)}$ | $2^{a-1}-1$ |
| No. of mod. exp. | - | - | 1 | 1 |

Table 3.1: Comparison of methods of Naor et al. and Asano

Asano described how to reduce this overhead by using a representation for the primes. A prime $p_{i,B}$ corresponding to $\gamma(i,B)$ is defined to be the $(B)_2^{th}$ smallest prime larger than $(i-1)X$, where $(B)_2$ denotes a binary number represented by a bit string $B$ and $X$ is a positive integer. If $X$ is chosen large enough, then each interval $((i-1)X, iX]$ will contain the required $2^a-1$ primes. This adds $\mathcal{O}\left(\frac{2^a\log^5(n)}{\log(a)}\right)$ to the computational complexity (including primality testing), but the receiver needs only store $X$, which is roughly $(2^a-1)\ln(2^a n\log(2^a n))$ in size.

### 3.2.2 Compression Method 2

The second Method of Asano uses the same ideas, only in a more watered-down manner. Instead of only one Master Key, each user will be given a Master Key for each node on the path to the root. This relaxes the computational expense, and only slightly increases the storage required. The centre needs only choose $2^a-1$ primes $p_B$ (made public), but must choose $\frac{n-1}{a-1}$ random numbers $K_i$ mod $M$ and set:

$$L_{i,B} = K_i^{T/p_B} \mod M,$$

where $T=\prod p_B$ is the product of all the primes. The Master Keys are indexed by both the user $u_j$, and the node on the tree $v_i$, and are calculated from:

$$MK_{i,j} = K_i^{T/w_{i,j}} \mod M,$$

[1]These formulae are upper bounds

where $w_{i,j}$ is the product of all primes $p_B$ where $u_j \in \gamma(i, B)$. To generate any key $L_{i,B}$, the user $u_j$ must take his Master Key for that node on the tree $MK_{i,j}$, and get the product of all primes for all subsets except for $B$. This being $w_{i,j}/p_B$, they work out:

$$MK_{i,j}^{w_{i,j}/p_B} \mod M = \left(K_i^{T/w_{i,j}}\right)^{w_{i,j}/p_B} \mod M = K_i^{T/p_B} \mod M = L_{i,B}.$$

Since a user can only belongs to $2^{a-1} - 2$ subsets (or $2^{a-1} - 1$ for the root), the user has much less multiplications compared to Method 1. Because there are fewer primes, they can be smaller and will take less space to store. The disadvantage of Method 2 over Method 1 is the user is storing $\log_a(n)$ Master Keys instead of just 1.

In Chapter 4 we will take a closer look at $t_{\max}(n, r)$ as well as $t_{aver}(n, r)$ for the Complete Subtree based on an $a$-ary tree. We will also propose a third alternative for reducing the storage needed.

## 3.3 Subset Difference Revocation Scheme

This is the second scheme proposed by Naor, Naor and Lotspiech, and in this section we define the scheme, as well as give the cover algorithm from [23]. Like the Complete Subtree on an $a$-ary tree, the scheme does require the user have considerably more keys than they would with the basic Complete Subtree. We will describe the method of Naor et al. to reduce the storage. Although it has the same principle as the Master Key method of Asano, it is very different (uses a Pseudo Random Number Generator rather than RSA calculations).

The Subset Difference Revocation Scheme is another tree-based Revocation Scheme, with a lot of similarities to the Complete Subtree Revocation Scheme. As before we will use a binary tree to define the subsets of users who share the establishment keys. Whereas with the Complete Subtree Revocation Scheme (and as we will see later with the Forest of Trees Revocation Scheme) we assigned keys (or rather indices of keys) to single nodes, this time we will assign them to pairs of nodes. The tree is a complete binary tree with $2n - 1$ nodes, and we will use breadth first labelling of the nodes, $v_i,\ i = 1 \ldots 2n - 1$. The index set $\Omega$ is a set of pairs of such indices, $(i, j)$. The index set does not

contain every pair of indices, but rather we say that the index pair $(i, j)$ is in $\Omega$ provided the node $v_i$ is an ancestor of the node $v_j$ (i.e. $v_i$ is on the path from $v_j$ to the root). The key corresponding to the index pair $(i, j)$ is $L_{i,j}$.

Note that with breadth first labelling, the leaves of the tree are labelled $v_n, \ldots, v_{2n-1}$. As with the Complete Subtree Revocation Scheme, each user will be assigned to one of these leaves. The subset of users $\gamma(i, j)$ (i.e. all the users who will have a copy of the key $L_{i,j}$) will be obtained from the difference between all the descendants of $v_i$ and all the descendants of $v_j$. Since $v_i$ is an ancestor of $v_j$, this is just the descendants of $v_i$ who are not descendants of $v_j$.

*Definition* 15. A Subset Difference Revocation Scheme $(\mathcal{N}, \Omega, \gamma_f)$ on $n = 2^k$ users is defined as follows. Let $T$ be a complete binary tree with $2n-1$ nodes, $\{v_1(= root), \ldots, v_{2n-1}\}$, indexed using breadth first labelling. Let $desc(v_i)$ for $i \in \{1, \ldots, 2n-1\}$ denote the subset of leaves that are descendants of the node $v_i$.

$$\begin{aligned} \mathcal{N} &= \{u_1, \ldots, u_n\} \\ \Omega &= \{(i, j) : (v_i, v_j \in T \text{ and } desc(v_j) \subset desc(v_i)) \text{ or } (i, j) = (0, 0)\} \end{aligned}$$

Each index pair $(i, j) \in \Omega$ corresponds to two nodes in the tree $T$, $(v_i, v_j)$, with $v_j$ being descended from $v_i$. Let $f$ be a bijection that maps leaves to users:

$$f : [v_n, \ldots, v_{2n-1}] \rightarrow \mathcal{N}.$$

We define $\gamma$ of any index pair $(i, j)$ in terms of the sets of descendants of the nodes of $T$ corresponding to $i$ and $j$:

$$\gamma(i, j) = \begin{cases} \{f(l) : l \in desc(root)\} & \text{if } (i, j) = (0, 0) \\ \{f(l) : l \in desc(v_i) \setminus desc(v_j)\} & \text{otherwise.} \end{cases}$$

The condition that $desc(v_j) \subset desc(v_i)$ guarantees that $v_j$ is descended from $v_i$. If $desc(v_i)$ and $desc(v_j)$ have any one leaf in common, then the path from that leaf to the root must pass through both $v_i$ and $v_j$ (by the definition of $desc(v)$). So the descendants of whichever node is the closer to the root will contain the lower node's descendants. We will use this fact in Chapter 4, when we show that a minimal cover in the Complete Subtree Revocation Scheme is disjoint.

As we have said, not all pairs of indices are present in $\Omega$. For any $(i, j) \in \Omega$, $v_j$ must be a node on the tree $T$ descended from $v_i$ (ignoring the pair $(0, 0)$ for the moment). This means that $v_i$ must be at least one level above the leaves. So the set of indices $\{i : (i, j) \in \Omega\}$ is just $\{1, \ldots, n-1\}$. This is all the nodes with the exception of the leaves. If we fix $i$, and say that $v_i$ is at height $h$, then the corresponding set for $j$, $\{j : (i, j) \in \Omega\}$, is just $\{2^{h'}i + j' : h' = 1, \ldots, h, j' = 0, \ldots, 2^{h'} - 1\}$, since we are using breadth first labelling. The extra index pair, $(0, 0)$, is added so that we can make the following claim:

*Lemma* 16. Let $(\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme, and let $(\mathcal{N}, \Omega', \gamma')$ be a Subset Difference Revocation Scheme on the same set of users. If the same tree was used to define both methods, we have:

$$\{\gamma(i) : i \in \Omega\} \subseteq \{\gamma'(i', j') : (i', j') \in \Omega'\}.$$

*Proof.* For the Complete Subtree Revocation Scheme, $\Omega$ is comprised of one index for each node on the tree. With the one exception of the root, all nodes have a parent. Consider any node that is not the root. As we use breadth first labelling, if the parent node is $v_i$, the left child is node $v_{2i}$, and it's sibling, the right child, is $v_{2i+1}$. The indices are $i$, $2i$ and $2i + 1$, respectively. The output of $\gamma(2i)$ ($CSRS$) is simply the set of users corresponding to the leaves that are descended from $v_{2i}$.

Consider the output of $\gamma'(i, 2i+1)$ ($SDRS$). Since the same tree was used, this set of users corresponds to those leaves that are descendants of $v_i$, but not of $v_{2i+1}$. But in a binary tree all leaves descended from $v_i$ must be descended from one of its two children. If we remove all descendants of $v_{2i+1}$, we are left with only those leaves that are descended from $v_{2i}$:

$$\begin{aligned} desc(v_i) &= desc(v_{2i}) \cup desc(v_{2i+1}) \\ \text{which implies} \quad desc(v_{2i}) &= desc(v_i) \setminus desc(v_{2i+1}). \end{aligned}$$

So by the definition of $\gamma'$, we have that:

$$\gamma(2i) = \gamma'(i, 2i + 1).$$

The exact same argument holds if we picked a right child ($\gamma(2i+1) = \gamma'(i, 2i)$). We cannot apply the same to $\gamma(root)$, since the root has no parent. So the

| | **Cover Algorithm** |
|---|---|
| 0: | Initialise $S = \{\}, T' = ST(\mathcal{R})$ |
| 1: | **while** $T'$ has more than one leaf **do** |
| 2: | Find two leaves in $T'$, $v_i$ and $v_j$, such that their least common ancestor, $v$, has no other descendants in $T'$. |
| 3: | Let $v_k$ be the child of $v$ such that it is an ancestor of $v_i$ and $v_l$ be the child of $v$ such that it is an ancestor of $v_j$. |
| 4: | **if** $v_i \neq v_k$ (i.e. $v_i$ is not a child of $v$) **then** $S = S \cup \{\gamma(k, i)\}$. |
| 5: | **if** $v_j \neq v_l$ **then** $S = S \cup \{\gamma(l, j)\}$. |
| 6: | Remove everything descended from $v$ from the tree $T'$, leaving $v$ a revoked leaf. This is the updated $T'$. |
| 7: | **end do** |
| 8: | **if** the remaining leaf $v_i$ is not the root **then** $S = S \cup \{\gamma(root, i)\}$. |

Table 3.2: Algorithm to find the cover in the Subset Difference Revocation Scheme.

above does not apply directly to the Subset Difference Revocation Scheme. But $\gamma(root) = \mathcal{N}$, as all leaves are descended from the root. Since we have that $\gamma'(0,0) = \mathcal{N}$ with $(0,0) \in \Omega'$ the statement holds for all $i \in \Omega$. □

This also means we have the singletons for every leaf. If $v_i$ is a leaf, then the user assigned to that leaf is $f(v_i)$, and corresponding singleton is:

$$\{f(v_i)\} = \gamma(par(v_i), sib(v_i)).$$

This satisfies the requirement of Theorem 4 and so this scheme is a Revocation Scheme.

Naturally, the algorithm for finding the cover of $\mathcal{N} \setminus \mathcal{R}$ is very different to that for the Complete Subtree Revocation Scheme. We do start off with a Steiner Tree $ST(\mathcal{R})$ connecting all revoked users and the root, as before. The algorithm described in [23] adds subsets to the cover one or two at a time while pruning $ST(\mathcal{R})$. Table 3.2 contains a description of this algorithm to find the cover if $r \neq 0$.

The subtree $T'$ gets pruned each pass through the algorithm, but the original tree $T$ from the definition of the scheme stays the same. The function of the tree $T'$ is to determine which nodes of $T$ we need to use as indices to find the cover. In order to find the least common ancestor in Step 2, we use the

algorithm from Lemma 11. This does not require that the subtree be complete, only that it is finite. The output of the algorithm is the set $S$, which is a cover of all privileged users.

Each iteration through the while loop reduces the number of revoked leaves in $T'$ by 1 and increases the size of the set $S$ by at most 2. This happens all the way until $T'$ has just one leaf, when the last step may add one more subset. Therefore we can state that:

*Corollary* 17. Let $(\mathcal{N}, \Omega, \gamma)$ be a Subset Difference Revocation Scheme, with $n$ users. For any $1 \leq r \leq n$:

$$t_{\max}(n, r) \leq 2r - 1.$$

When $r = 0$, then we can not actually use this algorithm, since there is no $ST(\mathcal{R})$. But we did add an index to the scheme for this specific case $(\gamma(0, 0) = \mathcal{N})$, so $t_{\max}(n, 0) = 1$.

### 3.3.1 Storage

We now show that the storage requirement of each user is (roughly) $4n$ in the Subset Difference Revocation Scheme, as compared to $\log_2(n) + 1$ with Complete Subtree Revocation Scheme. This idea was sketched in [23] and we complete it here. We look at it from the point of view of a single user $u$ and try to classify all the keys he/she has (the set $U$). Let's say user $u$ is assigned leaf $v$ on the tree. This user will have the key $L_{i,j}$ provided that $v_i$ is on the path from $v$ to the root and $v_j$ is descended from $v_i$, but not on the path from $v$ to the $v_i$. Looking at the subtree rooted at $v_i$, and assuming $v_i$ is at a height $h$, then there are $2^{h+1} - 1$ nodes in this subtree. Out of these, $h+1$ are on the path from the leaf $v$ to $v_i$. So there are only $2^{h+1} - 1 - (h + 1)$ possibilities for $v_j$, such that user $u$ holds key $L_{i,j}$. Since $n = 2^k$, and the scheme is based on a complete binary tree we have $k$ edges from any leaf to the root. We have already shown that for any index pair $(i, j) \in \Omega$, $v_i$ cannot be a leaf. So we

sum over the $k$ nodes from $v$'s parent to the root:

$$\begin{aligned}
|U| &= \sum_{h=1}^{k} 2^{h+1} - 1 - (h+1) \\
&= \sum_{h=1}^{k} 2^{h+1} - h - 2 \\
&= \sum_{h=1}^{k} 2^{h+1} - \sum_{h=1}^{k} h - \sum_{h=1}^{k} 2 \\
&= (2^{k+2} - 4) - \frac{k(k+1)}{2} - 2k \\
&= 4.2^k - \frac{k^2}{2} - \frac{5k}{2} - 4 \\
\text{So} \qquad |U| &= 4n - \log_2^2(n)/2 - 5\log_2(n)/2 - 4.
\end{aligned}$$

We can get an idea on the improvement of the Subset Difference Revocation Scheme by comparing the upper bounds on $t_{\max}(n, r)$. For the Complete Subtree Revocation Scheme we had that $t_{\max}(n, r) \leq r \log_2(n/r)$. From looking at the algorithm to find the cover we found that $t_{\max}(n, r) \leq 2r - 1$ when using $SDRS$. Also, by Lemma 16, we know that any cover found using $CSRS$ can be replicated with $SDRS$. This means that $t_{\max}(n, r) \leq n \log_2(n/r)$ applies to both methods. So we can say that in terms of bandwidth, $SDRS$ performs better (for small values of $r$, we have $2r - 1 < r \log_2(n/r)$). The tradeoff for this improvement lies in the large number of keys each user has to store. A storage requirement of just under $4n$ is undesirable given that we want to be able to have a large population of users. $CSRS$ only requires $\log_2(n) + 1$ keys to be stored by each user. However, we can reduce the storage needed in $SDRS$ by sacrificing information-theoretic security for computational security.

### 3.3.2 Pseudo Random Sequence Generator

Rather than having the keys chosen uniformly at random and independently from each other, Naor et at. ([23]) suggested that they be generated so that the users can use some secret information (or labels) to calculate the keys. They described the following method where each index pair $(i, j) \in \Omega$ is associated with a label ($LABEL_{i,j}$), as well as associating a key. The idea is that for

any index pair $(i, j)$, for which a user has a label, that user will be able to calculate all labels and keys for the index pairs $(i, j')$, where $v_{j'}$ is a descendant of $v_j$ (including the key $L_{i,j}$). It should be infeasible for anyone to reverse the calculation, i.e. calculate $LABEL_{i,j}$ from $LABEL_{i,j'}$ where $v_{j'}$ is a descendant of $v_j$. This could give someone the key corresponding to a subset to which they do not belong. This is all accomplished with the use of a Pseudo Random Sequence Generator.

*Definition* 18. A function $G : \{0,1\}^a \mapsto \{0,1\}^b$ is a *Pseudo Random Sequence Generator* if no polynomial-time adversary can distinguish the output of $G$ on a randomly chosen input string from a truly random string of similar length.

We call the input to the generator the *seed*, and typically the output is longer than the input. For our purposes, we will need the generator to triple the length of the input. There are two important consequences of the definition.

If an adversary cannot distinguish between the output of the generator and a random string, then he cannot invert the generator (in polynomial time). Suppose he did have the ability to retrieve the input from any string that was the output of a Pseudo Random Sequence Generator. Given two strings, of which only one is the output from the generator, he could try to invert both of them. Since only one was the output of the generator, only one will return a corresponding input string. This gives him the ability to distinguish the output and a random string, which is infeasible by definition. As well as this, given part of the output he can not predict any other part of the output. This would also allow an adversary to distinguish between the output of the generator and a random string.

The users will run the generator with one of their labels as input, $LABEL_{i,j}$. The output will be in the following form:

$$G(LABEL_{i,j}) = LABEL_{i,2j}||L_{i,j}||LABEL_{i,2j+1},$$

where $||$ is the concatenation operator. The middle third is the key corresponding to the index pair $(i, j)$. The two outer thirds are labels for different index pairs. Since we are using breadth first labelling, the nodes indexed by $2j$ and $2j + 1$ are the children of the node indexed by $j$. But how do we know

the user can not generate a key they are not supposed to have? Call the user $u$, and the label they are given $LABEL_{i,j}$. They will only be given this label if $u \in \gamma(i,j)$. From the definition of the Subset Difference scheme, we have that $\gamma(i,j)$ is just the set of users whose leaves are in $desc(v_i) \setminus desc(v_j)$ ($v_i$ is the node indexed by $i$, $v_j$ the node indexed by $j$). By the hierarchial nature of a binary tree we have that $desc(v_{j'}) \subset desc(v_j)$, where $v'_j$ is any descendant of $j$. Therefore:

$$(desc(v_i) \setminus desc(v_j)) \subset (desc(v_i) \setminus desc(v_{j'})).$$

Consequently, if $u \in \gamma(i,j)$ that implies $u \in \gamma(i,j')$, for any index pair with $j'$ descended from $j$. And these are the only labels/keys that $u$ can generate with $LABEL_{i,j}$.

The centre generates $LABEL_{i,i}$ for each node on the tree (only need it for the root and internal nodes, not leaves). It then uses the sequence generator to generate the labels that the users need. Any user $u$ is given the labels $LABEL_{i,j}$ where the $i$'s are the nodes on the path from the root to $u$ and $j$'s are the nodes that just "hang off" the path from $i$ to $v$ (the leaf user $u$ is assigned to). All keys that $u$ could need can be generated from these (and none that $u$ should not have since none of the $j$'s are ancestors of $v$ and all the $i$'s are). So if $i$ is at height $h$ then there are $h$ different $j$'s, so the total number of labels held by any user is:

$$1 + \sum_{i=1}^{\log_2 n} i = 1 + \frac{(\log_2(n)+1)\log_2(n)}{2} = \frac{1}{2}\log_2^2(n) + \frac{1}{2}\log_2(n) + 1. \qquad (3.5)$$

This results in a Revocation Scheme which is the same in terms of keys that are used to encrypt any broadcast, but differs in what is generated by the centre on initialisation and in what is stored by the users. So we have the exact same bound on the bandwidth, but the amount of storage needed by any one user is now of the order $\mathcal{O}(\log^2(n))$ instead of $\mathcal{O}(n)$.

Note that any user will only have to run the pseudo-random sequence generator at most $\log_2(n)$ times to generate a key from a label. The greatest separation between a label a user has and the label a user needs would occur when $v_j$ is a child of the root and $v_{j'}$ is a leaf. This would require $\log_2(n) - 1$

executions of the generator to traverse, plus one extra to get the key from the label.

This section has mostly been a summary of [23]. In Chapter 6 we will present our results on these scheme. These are focused on the communication costs of this scheme: we improve on the bound of $t_{\max}(n, r) \leq 2r - 1$ and investigate $t_{aver}(n, r)$.

# Chapter 4

# Complete Subtree Revocation Scheme

In this chapter we will derive the formula for $t_{\max}(n, r)$ for the Complete Subtree Revocation Scheme, improving on the bound of Naor et al. in [23]. This allows us to calculate exactly the maximum bandwidth given the number of revoked users. We will also show how to calculate $t_{aver}(n, r)$, which gives us a different perspective on the bandwidth of the scheme. In the later sections, we show how these measures are easily generalised to the $a$-ary tree variety. We also provide a third Compression Method that compliments the existing two of Asano [1].

We begin with a quick review of the Complete Subtree Revocation Scheme described in Chapter 3. $\mathcal{N}$ is the set of users ($|\mathcal{N}| = n$), $\Omega$ is the index set (each index is a placeholder for an establishment key), $T$ is a complete binary tree with $n$ leaves and the functions $\gamma$ and $\delta$ determine what users get which keys. From the user's point of view, we have:

- User $u$ is assigned a leaf, $v_j$, on a complete binary tree $T$ ($f(v_j) = u$).
- $\delta(u)$ corresponds to the keys $u$ is given
- $\delta(u) = \{i : v_i \in path \text{ from } v_j \text{ to the } root\}$

And for a key $L_i$:

- Index in $\Omega$ for this key is $i$

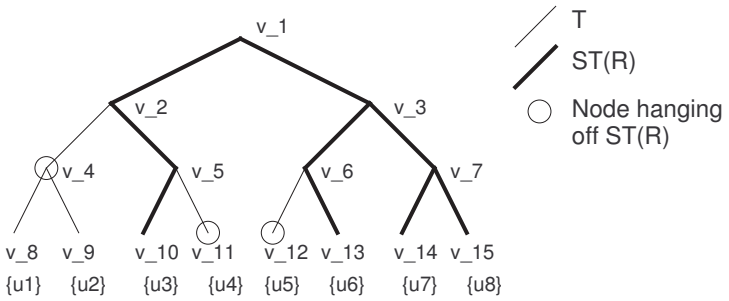


Figure 4.1: Example of the Complete Subtree Revocation Scheme.

- Node on tree assigned to this index is $v_i$
- $\gamma(i)$ is the set of users who are given key $L_i$
- $\gamma(i) = \{u : u\text{'s leaf is descended from } v_i\}$

In order to find a cover of any $\mathcal{N}\backslash\mathcal{R}$, all the centre does is find all nodes that hang off $ST(\mathcal{R})$. The cover is the partition of $\mathcal{N} \setminus \mathcal{R}$ consisting of the subsets of users that correspond to the subsets of leaves descendant from the nodes that hang off $ST(\mathcal{R})$. This was proved in Lemma 13, and we demonstrate the fact in the following example:

*Example* 19. Let $CSRS$ be a Complete Subtree Revocation Scheme on $n = 8$ users. The 8 users are assigned to the leaves of a complete binary tree, indexed using breadth first labelling, as in Figure 4.1. If $\mathcal{R} = \{3, 6, 7, 8\}$, then the cover of $\mathcal{N} \setminus \mathcal{R}$ is $\{\gamma(4), \gamma(11), \gamma(12)\}$, since $v_4$, $v_{11}$ and $v_{12}$ are the only nodes that hang off $ST(\mathcal{R})$.

The subtree $ST(\mathcal{R})$ is unique, and therefore the cover that you will get by the above process is unique. However, that does not mean that there is only one cover for any set of revoked users. On the contrary, if any one node $(v)$ we use is not a leaf, then we can get the same cover by using the keys corresponding to the two children of $v$ instead of $v$, along with the rest of the nodes. There is no point in doing this as it increases $t(\mathcal{N}, \mathcal{R})$ for no good reason, but it shows that the cover is almost certainly not unique. Another point about the cover is that it is disjoint.

*Lemma* 20. Let $(\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme. Let $C$ be a cover for any $\mathcal{N} \setminus \mathcal{R}$. Then $C$ is a disjoint cover.

*Proof.* Suppose the cover $C$ has two sets $\gamma(i)$ and $\gamma(j)$ that have users in common. This corresponds to the nodes on the tree, $v_i$ and $v_j$, having descendant leaves in common. The path from any of these leaves to the root must go through both $v_i$ and $v_j$, by the definition of a descendant. This means that one of the nodes is descended from the other. But all the nodes we use to generate the cover are roots of distinct subtrees hanging off $ST(\mathcal{R})$. Since each subtree can only have one root, we do not get a node that is descended from another in the cover. Therefore the cover is disjoint. □

If we were to allow overlapping sets, then one set would be contained in the other because of the hierarchical nature of the binary tree. This would just add needless redundancy.

## 4.1 Maximum Bandwidth

We now try to evaluate the performance of this scheme. The number of keys stored by each user is fixed at $\log_2(n) + 1$. As for the bandwidth, we will first consider the size of the maximum bandwidth: $t_{\max}(n, r)$. We have already have that $t_{\max}(n, r) \leq r \log_2(n/r)$ in Lemma 101. In this section we will derive an exact formula for $t_{\max}(n, r)$ for the Complete Subtree Revocation Scheme. In order to do this, we will classify all $ST(\mathcal{R})$ such that $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$, starting with the simple case when $r = 2$.

First, we can add a bound on the value of $t_{\max}(n, r)$, which applies to most tree-based schemes:

*Lemma* 21. Let $CSRS = (\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme with $n = 2^k$ users. Then $CSRS$ has:

$$t_{\max}(n, r) \leq n/2.$$

*Proof.* This bound is evident if we partition the set of users as follows:

$$\mathcal{N} = \{u_1, \ldots, u_n\} = \bigcup_{i=0}^{\frac{n}{2}-1} \{u_{2i+1}, u_{2i+2}\}.$$

For any subset $\mathcal{P} = \mathcal{N} \setminus \mathcal{R}$, $\mathcal{P} \subseteq \mathcal{N}$, we have that:

$$\mathcal{P} \cap \left\{ \bigcup_{i=0}^{\frac{n}{2}-1} \{u_{2i+1}, u_{2i+2}\} \right\} = \mathcal{P} \cap \mathcal{N} = \mathcal{P}.$$

Therefore, the sets $\mathcal{P} \cap \{u_{2i+1}, u_{2i+2}\}$, for $i = 0, \ldots, n/2 - 1$, form a cover. To show that it forms a cover using only subsets from the Complete Subtree Revocation Scheme, we need to show that $\mathcal{P} \cap \{u_{2i+1}, u_{2i+2}\}$ is either empty or equal to $\gamma(j)$, $j \in \Omega$, for any $i = 0, \ldots, n/2 - 1$. If $\mathcal{P} \cap \{u_{2i+1}, u_{2i+2}\}$ equals any of $\{u_{2i+1}\}, \{u_{2i+2}\}, \emptyset$, then this is true. From Theorem 4 we have that $\{u\} \in S$, for all $u \in \mathcal{N}$ (where $S$ is the range of $\gamma$).

We must show that $\{u_{2i+1}, u_{2i+2}\} \in S$ also. Consider the value $\gamma(n/2 + i)$. To evaluate the function, we first need the descendants of $v_{n/2+i}$. Since $n/2 + i < n$ (the label of the leftmost leaf), $v_{n/2+i}$ has descendants, which are $\{v_{n+2i}, v_{n+2i+1}\}$. Applying $f$ to these nodes we get $\gamma(n/2+i) = \{u_{2i+1}, u_{2i+2}\}$. Clearly $n/2+i \in \Omega$, as $\Omega$ is just $\{1, \ldots, 2n-1\}$ and $n/2+i < n/2+n/2-1 = n-1$. Therefore, $\{u_{2i+1}, u_{2i+2}\} \in S$ and the subsets $\mathcal{P} \cap \{u_{2i+1}, u_{2i+2}\}$ (for $i = 0, \ldots, n/2-1$) form a cover in the Complete Subtree Revocation Scheme. Because the range of $i$ is $0, \ldots, n/2 - 1$, there can be at most $n/2$ subsets in the cover. Therefore:

$$t_{\max}(n, r) \leq \frac{n}{2}. \qquad \square$$

*Corollary* 22. Let $CSRS = (\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme with $n = 2^k$ users. Then $CSRS$ has:

$$t_{\max}(n, r) \leq \min(\lfloor r \log_2(n/r) \rfloor, n - r, n/2).$$

*Proof.* From [23] (and Lemma 101) we know that $t_{\max}(n, r) \leq r \log_2(n/r)$. Since $t(\mathcal{N}, \mathcal{R})$ must always be a whole number, $t_{\max}(n, r) \leq \lfloor r \log_2(n/r) \rfloor$. From Corollary 6 we know that $t_{\max}(n, r) \leq n - r$. The third bound is from Lemma 21, which we just proved. $\square$

The bound in Lemma 21 cannot be extended any further. The subsets $\mathcal{P} \cap \{4i + 1, 4i + 2, 4i + 3, 4i + 4\}$ do form a cover, but not using subsets from the Complete Subtree Revocation scheme. If the size of the intersection is 3,

| $r$ | 0 | $1 \leq r < \frac{n}{4}$ | $r = \frac{n}{4}$ |
|---|---|---|---|
| Lowest Bound | 1 | $\lfloor r \log_2(\frac{n}{r}) \rfloor$ | $\lfloor r \log_2(\frac{n}{r}) \rfloor = \frac{n}{2}$ |
| $r$ | $\frac{n}{4} < r < \frac{n}{2}$ | $r = \frac{n}{2}$ | $\frac{n}{2} < r \leq n$ |
| Lowest Bound | $\frac{n}{2}$ | all 3 equal | $n - r$ |

Table 4.1: Lowest of three bounds of $t_{\max}(n, r)$ for the different ranges of $r$.

| n | r | $t_{\max}(n, r)$ | $\min(\lfloor r \log_2(n/r) \rfloor, n - r, n/2) =$ |
|---|---|---|---|
| $2^5$ | 19 | 13 | min( 14, 16, 13 ) = 13 |
| $2^6$ | 19 | 32 | min( 33, 45, 32 ) = 32 |
| $2^7$ | 19 | 51 | min( 52, 109, 64 ) = 52 |

Table 4.2: Differences between the bounds and $t_{\max}(n, r)$

then the subset will not equal $\gamma(i)$ for any $i \in \Omega$. The total number of users who share any key is always a power of 2 when we use a complete binary tree.

In Table 4.1 we see which of the three bounds are the lowest for the different values of $r$. The original bound of Naor et al. was not tight, we do not have to look at very large values of $n$ before we find a counter-example. For $n = 2^5$ and $r = 19$ we get $t_{\max}(n, r) = 13$ and $\lfloor r \log_2(n/r) \rfloor = 14$. But the improved bound is not tight either.

In Table 4.2 we present cases where the three bounds are not tight. The cases shown are those that occur for the smallest values of $n$ and $r$. The values of $t_{\max}(n, r)$ are found by experimentation and examples of $ST(\mathcal{R})$ can be found in Appendix B. The first row in Table 4.2 represents the smallest example of $t_{\max}(n, r) < \lfloor r \log_2(n/r) \rfloor$. Similarly, the second row is the smallest example of $t_{\max}(n, r) < \min(\lfloor r \log_2(n/r) \rfloor, n - r)$, and the last row is the smallest example of $t_{\max}(n, r) < \min(\lfloor r \log_2(n/r) \rfloor, n - r, n/2)$. These tables show the shortcomings of the bounds on $t_{\max}(n, r)$, even for small values of $n$. So before we proceed any further we will first try to find an exact formula for $t_{\max}(n, r)$.

As we are looking at the maximum value of $t(\mathcal{N}, \mathcal{R})$, we need to discover what circumstances must arise to give the largest possible cover. In other words, what does $ST(\mathcal{R})$ look like when $t(\mathcal{N}, \mathcal{R})$ is at a maximum, and what

choice of $\mathcal{R}$ gives this state?

To begin with, we will look at the case where $r$ is very small. When $r = 0$, we have $t(\mathcal{N}, \mathcal{R}) = 1$. The key for the root is shared by all users, and so is the only establishment key needed in the broadcast, $t_{\max}(n, 0) = 1$. When $r = 1$, without loss of generality, we can choose the first user to be revoked ($u_1$) to be the first leaf on the tree. $ST(\mathcal{R})$ is then just the one path from leaf to root, $\log_2(n) + 1$ nodes in total, which means $\log_2(n)$ nodes hanging off (the last node is a leaf, which can not have nodes hanging off). So $t_{\max}(n, 1) = \log_2(n)$.

We can prove a very simple formula for the case of $r = 2$, which will give us some insight into the shape of $ST(\mathcal{R})$.

*Lemma* 23. Let $(\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme with $n = 2^k$ users. Let $\mathcal{R}$ be a set of revoked users with $|\mathcal{R}| = 2$. Then $t(\mathcal{N}, \mathcal{R}) = k - 2 + h$, where $h$ is the height of the node where the two paths in $ST(\mathcal{R})$ meet.

*Proof.* To find $t(\mathcal{N}, \mathcal{R})$ we will count the number of nodes that hang off $ST(\mathcal{R})$. We have already shown that this is the size of the cover (Lemma 13). $ST(\mathcal{R})$ is formed of two paths starting at the leaves, until they meet at height $h$ (call this node $v$), and then it is just one path to the root. The height is the number of edges on the path from $v$ to any of the leaves (strictly speaking it is the maximum number, but in a complete tree all leaves are at the same depth). This means that there are $h + 1$ nodes from $v$ to any leaf (inclusive). There can be no nodes hanging off a leaf, nor can there be any hanging off $v$ as it has both children in $ST(\mathcal{R})$. So we are left with $h - 1$ nodes hanging off each path. Node $v$ is distance $k - h$ from the root (distance from any leaf to root is $k$), which means $k - h + 1$ nodes. The number of nodes hanging off the path from $v$ to the root is one less than the number of nodes, since the root can have a node hanging off but $v$ can not. This gives the formula for $t(\mathcal{N}, \mathcal{R})$:

$$\begin{aligned} t(\mathcal{N}, \mathcal{R}) &= (k - h) + 2(h - 1) \\ &= k + h - 2. \end{aligned}$$

□

So the higher up the two paths meet, the larger $t(\mathcal{N}, \mathcal{R})$ is. If the two paths meet at the root (height $h = k$), we get the highest possible size of a cover,

$t(\mathcal{N}, \mathcal{R}) = 2k - 2$. This suggests that for any value of $r$, the $ST(\mathcal{R})$ that gives the highest value of $t(\mathcal{N}, \mathcal{R})$ may be the one which has all forks/splits in the tree as high as possible.

What we will prove in the following theorems is that when $t(\mathcal{N}, \mathcal{R})$ attains the maximum, $t_{\max}(n, r)$, then $ST(\mathcal{R})$ only forks/splits at the top of the tree and only has non-forking paths at the bottom. In the following discussion, when we use the notion of the degree of a node, unless explicitly stated otherwise, we are talking about the number of edges connected to a node on $ST(\mathcal{R})$. The degree of a node in the original complete tree is a lot more straightforward, as all internal nodes have degree 3, the root has degree 2 and the leaves have degree 1. Since $ST(\mathcal{R})$ is a subtree of the complete tree then the degree of a node in $ST(\mathcal{R})$ is less than or equal to the degree in the original complete tree.

We use the same technique in the following two proofs. We assume there is some subset $\mathcal{R}$ with $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$, but with $ST(\mathcal{R})$ that does not have the desired property. We show that there exists a slightly different subset $\mathcal{R}'$ of the same size, but has a larger cover. This contradiction shows that $\mathcal{R}$ must have the desired property.

*Lemma* 24. Let $\mathcal{R}$ be such that $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ in a Complete Subtree Revocation Scheme. Then for any internal node $v$ with the same degree in both $ST(\mathcal{R})$ and the complete binary tree, every node between $v$ and the root also has the same degree in $ST(\mathcal{R})$ and the complete binary tree.

*Proof.* $ST(\mathcal{R})$ is a connected tree comprised of the paths of the leaves in $\mathcal{R}$ to the root, and all paths to the root are unique. So for any node in $ST(\mathcal{R})$, the edge to its parent is also in $ST(\mathcal{R})$ (this is on the path to the root). If there is a difference between the degree of a node in the complete binary tree and $ST(\mathcal{R})$, then one of the edges from the node to one of its children must be absent from $ST(\mathcal{R})$. This gives us a node not in $ST(\mathcal{R})$ hanging off. If the degrees are the same in both $ST(\mathcal{R})$ and the complete tree, then either both children are in $ST(\mathcal{R})$ and no nodes hang off, or else the node is a leaf (which cannot have a node hanging off).

What we need to show is the following: If $ST(\mathcal{R})$ has an internal node $v$ with the same degree in both trees, but the same does not hold true for $v$'s

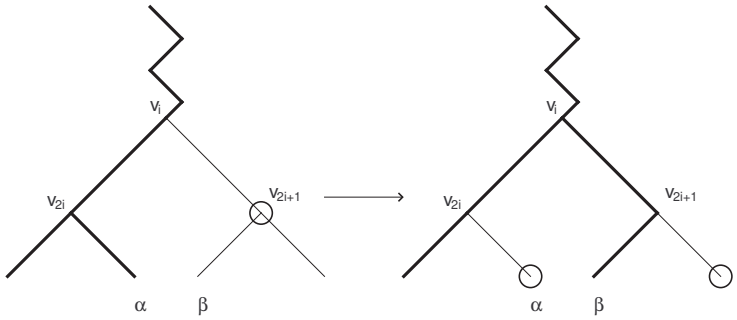


Figure 4.2: Part of a Complete Binary Tree ($ST(\mathcal{R})$ in thick lines).

parent, then we can create a subset $\mathcal{R}'$ such that $t(\mathcal{N}, \mathcal{R}) < t(\mathcal{N}, \mathcal{R}')$. This would imply that $t(\mathcal{N}, \mathcal{R}) < t_{\max}(n, r)$.

Let $\mathcal{R} \subseteq \mathcal{N}$ be a non-empty subset. Suppose $v_i$ is a node in $ST(\mathcal{R})$ with degree in $ST(\mathcal{R})$ strictly less than its degree in the original complete tree. Let one of its children be an internal node with the same degree in both trees (the other child must then be the one that hangs off). In calculating the size of any cover, we showed (Lemma 13) that what matters is the number of nodes that hang off and that their orientation (whether it is the left or right child that hangs off) does not matter. So, without loss of generality, we can choose the left child of $v_i$ ($v_{2i}$) to be in $ST(\mathcal{R})$, and the right child ($v_{2i+1}$) to hang off. Since $v_{2i}$ is an internal node, it must be at height at least 1. Because it has a parent, $v_i$, it cannot be the root either. So we have $1 \leq height(v_2) \leq \log_2(n) - 1$.

We will first consider the case where $v_{2i}$ is at height 1 ($v_{2i}$ is the parent of two leaves). Since the degree of $v_{2i}$ is 3, that means both paths out of $v_{2i}$ are in $ST(\mathcal{R})$. This means that the two leaves that are children of $v_{2i}$ are revoked. Let $\alpha$ be one of these leaves. The parent node $v_i$ has a node not in $ST(\mathcal{R})$ hanging off, which means $v_{2i}$'s sibling, $v_{2i+1}$, is not in $ST(\mathcal{R})$. So the two leaves that are descended from $v_{2i+1}$ are privileged. We will call one of these $\beta$. The size of the cover of all privileged leaves descended from $v_i$ is 1. There are only two privileged leaves and these are the only descendants of $v_{2i+1}$ (the one node that hangs off $ST(\mathcal{R})$). So the index for this node, $2i + 1$, is the only index needed for the cover.

Consider the set of leaves $\mathcal{R}' = (\mathcal{R} \setminus \{\alpha\}) \cup \{\beta\}$. It has the same cardinality

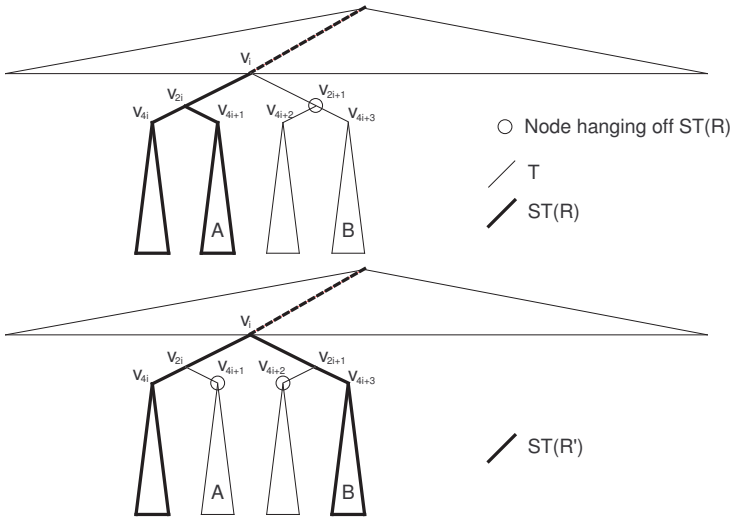


Figure 4.3: $ST(\mathcal{R})$ and $ST(\mathcal{R}')$ from Lemma 24.

as $\mathcal{R}$, since we have just removed one leaf and added another. But the cover has changed. Just looking at the four leaves descended from $v_i$, the number of nodes that hang off $ST(\mathcal{R})$ (in this small section of the tree) is just one, $v_{2i+1}$. In $ST(\mathcal{R}')$, there are two: $\alpha$ and $\beta$'s sibling (see Figure 4.2). In Lemma 13 we showed that the nodes hanging off $ST(\mathcal{R})$ form a minimal cover. Because the rest of the leaves in $\mathcal{R}'$ are the same as those in $\mathcal{R}$, the number of nodes hanging off the rest of $ST(\mathcal{R}')$ is the same as for $ST(\mathcal{R})$. Therefore:

$$\begin{aligned} t(\mathcal{N}, \mathcal{R}') &= t(\mathcal{N}, \mathcal{R}) + 1 \\ \text{i.e.} \quad t(\mathcal{N}, \mathcal{R}) &< t_{\max}(n, r). \end{aligned}$$

It is apparent that the cover of all privileged users in the subtree rooted at $v_i$ is independent of the rest of the tree. If any node that was an ancestor of $v_i$ (including $v_i$ itself) was included in the cover then the two revoked leaves would be included as well, since they are descended from $v_i$. The rest of the nodes outside the subtree rooted at $v_i$ cannot include any of these leaves because $v_i$ is not a descendant of any of them.

We now consider the more general case, where the height of $v_{2i}$ can be anywhere in the range $1 \leq height(v_{2i}) \leq \log_2(n) - 1$. The same argument holds, although we need to look at more of the tree. As before, we assume that $v_{2i}$ has the same degree in both $ST(\mathcal{R})$ and the complete tree, while $v_i$'s

degree in $ST(\mathcal{R})$ is less than that in the complete tree. The children of $v_i$ are $v_{2i}$ and $v_{2i+1}$. Their children are $v_{4i}, v_{4i+1}$ and $v_{4i+2}, v_{4i+3}$, respectively. Define $\mathcal{A}$ to be the set of leaves descended from $v_{4i+1}$ that are in $\mathcal{R}$. Since we have a complete binary tree, all nodes at the same level have the same number of leaves descended from them. This means that the subtrees rooted at $v_{4i+1}$ and $v_{4i+3}$ are identical (since $v_{4i+1}$ and $v_{4i+3}$ are at the same depth from the root). So we can define a set of leaves $\mathcal{B}$ to be a copy of the leaves in $\mathcal{A}$. In other words $\mathcal{B}$ is the set of leaves such that $ST(\mathcal{A})$ in the subtree rooted at $v_{4i+1}$ is exactly the same as $ST(\mathcal{B})$ rooted at $v_{4i+3}$. Because we are using breadth first labelling, we can define the set $\mathcal{B}$ explicitly. The labels for the two nodes $v_{4i+1}$ and $v_{4i+3}$ initially differ by 2. To get the label for the left child of each node, we just multiply by 2. The difference between these two labels ($v_{8i+2}$ and $v_{8i+6}$) is 4. Each level we go down, the difference between nodes is multiplied by two. So if we started with $v_{4i+1}$ at height $h$, the left most leaf descended from $v_{4i+1}$ will be $v_{2^h(4i+1)}$, and the left most leaf descended from $v_{4i+3}$ will be $v_{2^h(4i+3)}$, giving a difference of $2.2^h$. All similar descendants of the two nodes will have labels that differ by the same amount (labels increase by 1 as you go from left to right on the same level). This means the set $\mathcal{B}$ is:

$$\mathcal{B} = \{v_{j+2^{h+1}} | v_j \in \mathcal{A}\}, \qquad \text{where } h \text{ is the height of } v_{4i+1}.$$

If we define $\mathcal{R}' = (\mathcal{R} \setminus \mathcal{A}) \cup \mathcal{B}$, then it still has the same number of leaves as $\mathcal{R}$ ($\mathcal{A}$ and $\mathcal{B}$ are the same size). Just like before, there is a difference in the sizes of the covers. In the portion of the tree that is descended from $v_i$, to cover $\mathcal{R}$ there was one node hanging off $ST(\mathcal{R})$, $v_{2i+1}$, as well as anything in the subtrees rooted at $v_{4i}$ and $v_{4i+1}$. The subtree rooted at $v_{4i}$ is the same in both $ST(\mathcal{R})$ and $ST(\mathcal{R}')$. The subtree rooted at $v_{4i+1}$ in $ST(\mathcal{R})$ is the same as the subtree rooted at $v_{4i+3}$ in $ST(\mathcal{R}')$. On top of this now we have the nodes $v_{4i+1}$ and $v_{4i+2}$ both hanging off $ST(\mathcal{R}')$, instead of just the one ($v_{2i+1}$). Since the rest of the tree remains unchanged we have:

$$\begin{aligned} t(\mathcal{N}, \mathcal{R}') &= t(\mathcal{N}, \mathcal{R}) + 1 \\ \text{i.e.} \quad t(\mathcal{N}, \mathcal{R}) &< t_{\max}(n, r). \end{aligned}$$

This can be seen in Figure 4.3.

This proves that if $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ and $v$ is an internal node with the same degree in $ST(\mathcal{R})$ as in the original complete tree, then the same must be true of its parent. This argument can be iteratively applied to all nodes on the path to the root, including the root itself. □

Lemma 24 gives a necessary condition for $\mathcal{R}$ to give $t_{\max}(n, r)$, but it turns out that it is not sufficient. We need a generalisation of the above lemma, by showing that the same claim holds when $v_1$ is not the parent of $v_2$ (or even an ancestor of $v_2$).

*Theorem* 25. Let $\mathcal{R}$ be such that $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ using a Complete Subtree Revocation Scheme. Then for any internal node $v$ with the same degree in both $ST(\mathcal{R})$ and the complete binary tree, every node that is at a greater height than $v$ (i.e. closer to the root) also has the same degree in $ST(\mathcal{R})$ and the complete binary tree.

*Proof.* Assume that the statement of the theorem is false. Let $\mathcal{R}$ be such that $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ and suppose there exists two nodes $v_i$ and $v_j$ that contradict the theorem. Let $v_i$ be at height $h_i$ and have degree in $ST(\mathcal{R})$ less than its degree in the complete tree. Let $v_j$ be at height $h_j$, where $h_i > h_j$ ($h_i - h_j \geq 1$), and have the same degree in $ST(\mathcal{R})$ as in the complete tree. Let $v'_j$ be a child of $v_j$ (either one). Define $\mathcal{A}$ to be the set of all revoked leaves descended from $v'_j$. We need to choose a node $v'_i$ as follows: it must be descended from $v_i$, it must not be in $ST(\mathcal{R})$, and it must be at the same height as $v'_j$. Since $v_i$ only has one node not in $ST(\mathcal{R})$ hanging off, $v'_i$ must be descended from this node. We know the height difference between $v_i$ and $v_j$ is $h_i - h_j$, so the height difference between a child of $v_i$ (that is not in $ST(\mathcal{R})$), and a child of $v_j$, $v'_j$, is the same. So $v'_i$ can be any node that is at a depth of $h_i - h_j$ from the node hanging off $v_i$. There are $2^{h_i - h_j}$ nodes that satisfy this requirement, any of which will suffice. Let $\mathcal{B}$ be a set of leaves descended from $v'_i$, defined as follows: for every leaf $v_a$ in $\mathcal{A}$, there is a leaf $v_b$ in $\mathcal{B}$ such that the path from $v'_j$ to $v_a$ is the same as the path from $v'_i$ to $v_b$. This means the subtree $ST(\mathcal{A})$ rooted at $v'_j$ is the same as the subtree $ST(\mathcal{B})$ rooted at $v'_i$. Finally, define $\mathcal{R}' = (\mathcal{R} \setminus \mathcal{A}) \cup \mathcal{B}$.

How does the size of the cover of $\mathcal{N} \setminus \mathcal{R}'$ differ from that of $\mathcal{N} \setminus \mathcal{R}$? To

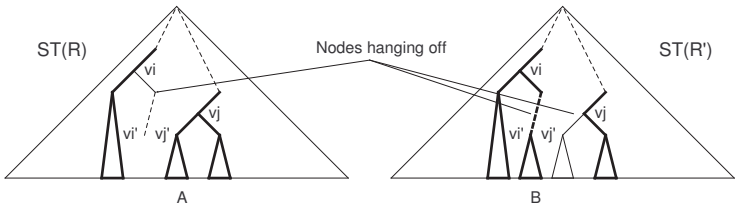


Figure 4.4: One node hanging off in $ST(\mathcal{R})$ and several in $ST(\mathcal{R}')$.

answer this all we need to know is where the two subtrees $ST(\mathcal{R})$ and $ST(\mathcal{R}')$ differ, and how many nodes hang off each tree in these locations. Since we created $\mathcal{R}'$ by removing the leaves $\mathcal{A}$ and adding the leaves $\mathcal{B}$, the differences in the two subtrees can only be due to the absence/presence of these leaves. In order to express $t(\mathcal{N}, \mathcal{R}')$ in terms of $t(\mathcal{N}, \mathcal{R})$, we need to make the following two observations:

1. As $\mathcal{A}$ is not present in $\mathcal{R}'$ (and is present in $\mathcal{R}$), instead of having the nodes that hang off $ST(\mathcal{A})$, $ST(\mathcal{R}')$ only has the one node hanging off $v_j$.

2. As $\mathcal{B}$ is present in $\mathcal{R}'$ (and not in $\mathcal{R}$), $ST(\mathcal{R}')$ does not have a node hanging off $v_i$ (by the definition of $\mathcal{B}$ this is where the paths from the leaves in $\mathcal{B}$ meet the rest of the tree), but instead has the nodes hanging off $ST(\mathcal{B})$ and the path from $v_i'$ to $v_i$.

The path from $v_i'$ to $v_i$ is length $h_i - h_j + 1$, but because any node hanging off $v_i'$ would be counted in $ST(\mathcal{B})$, there are only $h_i - h_j$ nodes hanging off. This gives:

$$\begin{aligned} t(\mathcal{N}, \mathcal{R}') &= t(\mathcal{N}, \mathcal{R}) - \text{ nodes hanging off } ST(\mathcal{A}) + 1 \\ &\quad + \text{nodes hanging off } ST(\mathcal{B}) - 1 + h_i - h_j \\ &= t(\mathcal{N}, \mathcal{R}) + h_i - h_j \\ &> t(\mathcal{N}, \mathcal{R}). \end{aligned}$$

This contradicts the assumption that $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. Therefore, no internal node with degree in $ST(\mathcal{R})$ less than its degree in the complete tree

can be at a greater height (closer to the root) than a node with the same degree in both trees. □

We can now find $\mathcal{R}$ that gives $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. Examples can be seen in Appendix B. In the following proof, we will show that all nodes with both children in $ST(\mathcal{R})$ (i.e. the same degree in both $ST(\mathcal{R})$ and the complete tree) must occur in the top of the tree. We will also show that the condition from Theorem 25 is sufficient to uniquely specify $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$.

*Corollary* 26. Let $CSRS = (\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme with $n = 2^k$ users. Let $r$ be an integer, $1 \leq r \leq n$, and let $j = \lfloor \log_2(r) \rfloor$. Then $CSRS$ has:

$$t_{\max}(n, r) = r(k - j) - 2(r - 2^j).$$

*Proof.* Let $\mathcal{R}$ be a subset of $\mathcal{N}$ such that $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. By Lemma 10, there are exactly $r - 1$ nodes in $ST(\mathcal{R})$ that have both children in $ST(\mathcal{R})$. How many nodes have depth less than or equal to $j$ in the complete binary tree $T$ (or equivalently how many nodes are at a distance $j$ or less from the root)? This is just the sum of the first $j$ powers of 2, $\sum_{i=0}^{j} 2^j = 2^{j+1} - 1$. There are $2^j - 1$ nodes with depth $\leq j - 1$.

Suppose $j = \log_2(r)$. This means $2^j = r$ and $2^j - 1 = r - 1$. So if every node with depth $\leq j - 1$ (and only these nodes) have both children in $ST(\mathcal{R})$, there will be $2^j - 1 = r - 1$ nodes, which gives $r$ leaves in $\mathcal{R}$. If any node at depth $j$ or greater had both children in $ST(\mathcal{R})$, a node at depth $\leq j - 1$ would only have one child in $ST(\mathcal{R})$ to ensure the number of leaves equals $r$. These two nodes would contradict Theorem 25, as we assumed the size of the cover is $t_{\max}(n, r)$. So for $j$ a power of two, we have that all nodes at depth $\leq j - 1$ have both children in $ST(\mathcal{R})$ (and no other nodes do). As a consequence of this, all nodes at depth $j$ are in $ST(\mathcal{R})$.

Alternatively, if $j < \log_2(r)$, then having all nodes at depth $\leq j - 1$ have both children in $ST(\mathcal{R})$ is not enough to give us $r$ leaves ($2^j - 1 < r - 1$). We must have at least one node at depth $j$ with both children in $ST(\mathcal{R})$. This rules out any node at depth $\leq j - 1$ having only one child in $ST(\mathcal{R})$ (Theorem 25). Neither could we have all nodes at depth $j$ with both children

in $ST(\mathcal{R})$, as this would imply that:

$$2^{j+1} - 1 \leq r - 1,$$

or

$$2^{j+1} \leq r.$$

But since $j = \lfloor \log_2(r) \rfloor$, $2^j$ is the greatest power of 2 less than or equal to $r$. So there is at least one node at depth $j$ with only one child in $ST(\mathcal{R})$. This node rules out any node with both children in $ST(\mathcal{R})$ at depth $> j$ (Theorem 25). So for $j$ not a power of two, we have that all nodes at depth $\leq j - 1$ and some at depth $j$ have both children in $ST(\mathcal{R})$. Since we need $r - 1$ nodes in total, and we have $2^j - 1$ from the nodes at depth $\leq j - 1$, this means that $(r - 1) - (2^j - 1) = r - 2^j$ nodes at depth $j$ have both children in $ST(\mathcal{R})$.

Consider the size of the cover of $\mathcal{N} \setminus \mathcal{R}$, namely the number of nodes that hang off $ST(\mathcal{R})$. The paths in $ST(\mathcal{R})$ that have nodes hanging off will start at either depth $j$ or $j + 1$. Since the path from the root to any leaf is length $k$, these paths will be of length $k - j$ and $k - j - 1$ respectively. The number of nodes that hang off is just the length of the path. All $r$ paths are at least $k - j - 1$ in length, giving $r(k - j - 1)$ nodes hanging off. The only extra nodes hanging off come from the fact that there some paths are longer. Since they are only 1 edge longer ($k - j$ versus $k - j - 1$), they only have one extra node hanging off. There are $r - 2^j$ nodes at depth $j$ with both children in $ST(\mathcal{R})$, so there remains $2^j - (r - 2^j) = 2^{j+1} - r$ nodes with only one child, and consequently a path of length $k - j$ to the leaves. So the size of the cover is:

$$\begin{aligned} t(\mathcal{N}, \mathcal{R}) &= r(k - j - 1) + 2^{j+1} - r, \\ &= r(k - j) - 2(r - 2^j). \end{aligned}$$

This value of $t(\mathcal{N}, \mathcal{R})$ is not effected by the choice of which $r - 2^j$ nodes picked at depth $j$ that have both children in $ST(\mathcal{R})$. The only parameter that can change the value of $t(\mathcal{N}, \mathcal{R})$ is the depth at which the nodes fork and by Theorem 25 this is the same for all $\mathcal{R}$ with $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. Therefore

$$t_{\max}(n, r) = r(k - j) - 2(r - 2^j). \tag{4.1}$$

□

Note the range of $r$ does not include $r = 0$ in Corollary 26. No power of two is less than or equal 0, so $j$ is undefined. If the centre does not want to revoke any users, then there is a key shared by all users (key associated with the root of the tree), $t(\mathcal{N}, \mathcal{R}) = 1$.

If $r$ is a power of 2 then this formula agrees with the original bound of Equation (3.2). Suppose $r = 2^j$, then the second part of the equation is zero, and since $k = \log_2(n), j = \log_2(r)$ we get:

$$t_{\max}(n, r) = r(k - j) - 2(r - 2^j) = r(\log_2(n) - \log_2(r)) = r \log_2(n/r).$$

We now know the exact value of $t_{\max}(n, r)$ for the Complete Subtree Revocation Scheme. We can see output from the formula in Figure 4.5, plotted against the bound of Naor et al, $r \log_2(n/r)$. The bound was close to the true formula of $t_{\max}(n, r)$, but by deriving the formula we have removed any uncertainty in this measure of bandwidth. This will be important in later chapters, when we wish to compare the performance of different schemes. We would only be able to make very limited statements about schemes if we just had bounds on performance measures to compare.

We have classified all possible $\mathcal{R}$ with $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. As a result of Theorem 25, the shape of $ST(\mathcal{R})$ from the root down is in the form of a complete binary tree, for as many levels as needed to make $r$ paths. The bottom part of $ST(\mathcal{R})$ is just single paths that do not split. There is a lot of freedom in the choice of $\mathcal{R}$. At the last level, where the nodes can still fork, different choice of nodes give the same size cover, and assuming $r$ is not a power of 2, we have a choice. The paths that descend from these nodes are free to go to any leaf descended from them. Some examples of such trees can be found in the figures in Appendix B.

As we have said, the bound of $r \log_2(n/r)$ coincides with $t_{\max}(n, r)$ whenever $r$ is a power of 2. From the above plot it seems that $t_{\max}(n, r)$ has constant slope between these points that coincide with $r \log_2(n/r)$. These points are the places where $j$ (the floor of $\log_2(r)$) goes up by 1, as $r$ increases. This is a consequence of Formula (4.1), which we see if we consider the difference

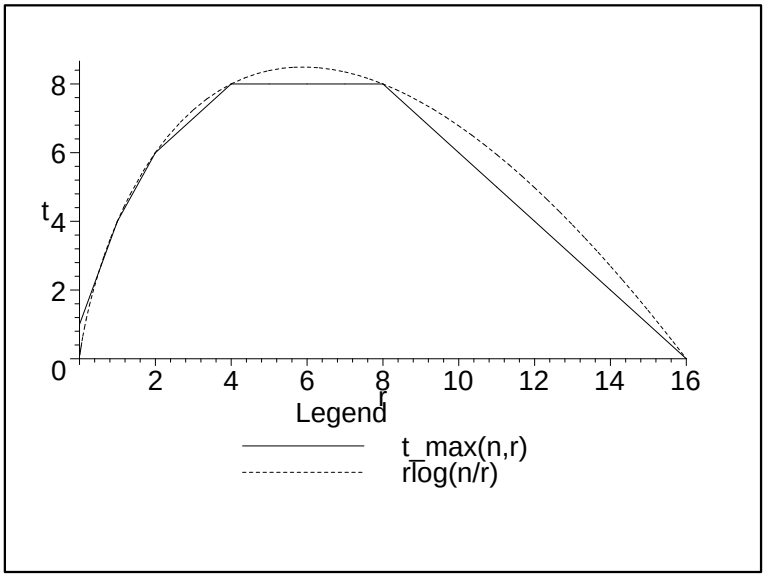


Figure 4.5: $t_{\max}(n, r)$ for the Complete Subtree Revocation Scheme (Formula (4.1)) and $r \log_2(n/r)$, when $n = 16$.

between consecutive values of $t_{\max}(n, r)$ when $j$ does not increase:

$$\begin{aligned} t_{\max}(n, r) &= r(k - j) - 2(r - 2^j) \\ t_{\max}(n, r + 1) &= (r + 1)(k - j) - 2(r + 1 - 2^j) \\ t_{\max}(n, r + 1) - t_{\max}(n, r) &= (k - j) - 2. \end{aligned}$$

Since this difference does not contain $r$, this increment does not change for any $r$ that gives $\lfloor \log_2(r) \rfloor = j$. So for $r$ in any range that has the same $j$, $t_{\max}(n, r)$ will increase by the same amount as $r$ increases, which gives us the straight line graph. These ranges are where $t_{\max}(n, r)$ is strictly less than the Formula (3.2). Where $r \log_2(n/r)$ follows a smooth curve between the points where $r$ is a power of 2, $t_{\max}(n, r)$ also goes through these points, but through a more direct route. Because the curve $r \log_2(n/r)$ always has decreasing slope (negative second derivative), the straight lines are below the curve. It is for this reason that we have equality for the bound in Lemma 21 in the range $n/4 \leq r \leq n/2$. Because $t_{\max}(n, n/4) = n/2$ and $t_{\max}(n, n/2) = n/2$, we must have $t_{\max}(n, r) = n/2$ for all values of $r$ in between. Similarly, $t_{\max}(n, r) = n - r$ for $n/2 \leq r \leq n$.

## 4.2 Average Bandwidth

In the last section, we were dealing with $t_{\max}(n, r)$. This was the formalising of an existing measure, and so there was already some work done on it. In this section we will look at our new measure of bandwidth, $t_{aver}(n, r)$, as calculated for the Complete Subtree Revocation Scheme. The formula (defined in Formula (2.2)) is:

$$t_{aver}(n, r) = \sum_{\substack{\mathcal{R} \subseteq \mathcal{N} \\ |\mathcal{R}|=r}} \frac{t(\mathcal{N}, \mathcal{R})}{\binom{n}{r}}.$$

There are two principle motivations investigating the average value of $t(\mathcal{N}, \mathcal{R})$. Firstly, the maximum $t(\mathcal{N}, \mathcal{R})$ is, by definition, an extreme event. It may not occur in the lifetime of any scheme. Second, it may be that the centre needs to measure the cost of the broadcasts in terms of the average length, rather than the maximum. For example, the cost of broadcasting data over the internet are typically calculated from the traffic sent per month. If the centre were to use this medium, then the scheme with the lowest average $t_{aver}(n, r)$ would be the most desirable, regardless of $t_{\max}(n, r)$.

In order to derive a formula for $t_{aver}(n, r)$, we will first find a more convenient way to sum over the range $\mathcal{R} \subseteq \mathcal{N},\ |\mathcal{R}| = r$. This allows us to express $t_{aver}(n, r)$ in terms of $t_{aver}(n/2, r')$, and so gives us a recursive relation.

We now say a few things about the average value of $t(\mathcal{N}, \mathcal{R})$, but first we will reiterate the notation. We denote by $t_{aver}(n, r)$ the average value of the size of the minimum cover over all possible subsets of $r$ revoked users from $\mathcal{N}$ using a Complete Subtree Revocation Scheme. For the extreme values of $r$, the size of the cover is constant, which makes the average trivial to calculate:

$t_{aver}(n, 0) = 1$ If no users are revoked, we use the key from the root to broadcast to everyone.

$t_{aver}(n, 1) = \log_2(n)$ To revoke one user, $ST(\mathcal{R})$ is just the one path, with $\log_2(n)$ nodes hanging off.

$t_{aver}(n, n) = 0$ If all users are revoked, we send nothing.

$t_{aver}(n, n-1) = 1$ If only one user is privileged, we only need the one key.

The last two cases are common to all Revocation Schemes, as they are consequences of Corollary 6. The first two are specific to the Complete Subtree Revocation Scheme. These are the few instances where the particular choice of the $r$ users does not affect the value of $t(\mathcal{N}, \mathcal{R})$, i.e. for any other values of $r$, different choices of revoked users can give different sized covers. Say we had $r = n-2$ (two privileged users). If the 2 privileged users were siblings then we can use their shared key for the broadcast, so $t(\mathcal{N}, \mathcal{R}) = 1$. Otherwise, they do not share a key and we have to use one for each of them, so $t(\mathcal{N}, \mathcal{R}) = 2$. The fact that the values for $t(\mathcal{N}, \mathcal{R})$ is always the same for these few values of $r$ (namely, $0, 1, n-1, n$) also means that the maximum coincides with the average.

We are going to use these "boundary values" to calculate other values of $t_{aver}(n, r)$ with the help of a recurrence relation. Firstly, we need to split up the $\binom{n}{r}$ subsets into different groupings (or a partition) indexed by a new parameter $i$. We will do this in such a way as to allow us to sum over $i$ when calculating $t_{aver}(n, r)$, instead of summing over $\mathcal{R}$.

*Definition* 27. An $(r, i)$-subset is a set $\mathcal{R}, |\mathcal{R}| = r$, of leaves on a complete binary tree with $n = 2^k$ leaves with the following property: the number of pairs of distinct sibling leaves $v_{j_1}, v_{j_2}$ such that both $v_{j_1} \in \mathcal{R}$ and $v_{j_2} \in \mathcal{R}$ is exactly $i$. We define $R_{(r,i)}(T)$ to be the set of all $(r, i)$-subsets on a complete binary tree $T$.

Consider any subset $\mathcal{R} \subseteq \mathcal{N},\ |\mathcal{R}| = r$. As the leaves of the complete tree are either revoked or privileged, one level up the parents of leaves fall into one of three categories: both children are revoked, both children are privileged or one child is revoked and one is privileged. We will later see why distinguishing between a revoked left/privileged right pair and revoked right/privileged left pair is unimportant. All we know about $\mathcal{R}$ is that is it a subset of size $r$ from the $n$ leaves in the tree. This is not enough to be able to say anything about which pairs of leaves are of which type. But the two parameters allow us to state two equations about the numbers of pairs of each type in $\mathcal{R}$:

$$|\text{revoked pairs}| + |\text{mixed pairs}| + |\text{privileged pairs}| = \frac{n}{2} \tag{4.2}$$

$$2 \times |\text{revoked pairs}| + |\text{mixed pairs}| = r. \tag{4.3}$$

Equation (4.2) comes from the fact that any complete binary tree with $n$ leaves has $n/2$ nodes one level up. Equation (4.3) is specific to the size of the subset $\mathcal{R}$. The number of revoked leaves in the revoked pairs set and mixed pairs set have to add up to $r$. If $\mathcal{R}$ is an $(r,i) - subset$, then we have that $|\text{revoked pairs}| = i$. This gives us two equations in two unknowns, that can be solved in terms of $n,\ r$ and $i$:

$$\begin{aligned} |\text{mixed pairs}| &= r - 2i \\ |\text{privileged pairs}| &= \frac{n}{2} - r + i. \end{aligned}$$

As $i$ will be the parameter we will be summing over to calculate $t_{aver}(n,r)$ instead of $\mathcal{R}$, we wish to know how many $(r,i) - subsets$ there are for any value of $i$. First we count the number of ways $n/2$ pairs of siblings can be partitioned into $i$ revoked pairs and $n/2 - r + i$ privileged pairs (with $r - 2i$ mixed pairs). This is simply the multinomial coefficient:

$$\binom{n/2}{(i)\ (r-2i)\ (n/2-r+i)} = \frac{(n/2)!}{(i)!(r-2i)!(n/2-r+i)!}.$$

Corresponding to each such partition, there are $2^{r-2i}$ subsets $\mathcal{R}$ with this distribution on pairs. This arises from the fact that the mixed pairs can each be one of two possibilities (privileged user to the left or to the right). This means that:

$$|R_{(r,i)}(T)| = \binom{n/2}{(i)\ (r-2i)\ (n/2-r+i)} 2^{r-2i}. \tag{4.4}$$

Of course, this formula assumes that $n$, $r$ and $i$ are such that there is at least one $(r,i) - subset$. We need to know the values of $i$ that can give $(r,i) - subsets$ for any particular $r$ and $n$. This is just the range of $i$ such that the three quantities $|\text{revoked pairs}|$, $|\text{mixed pairs}|$, $|\text{privileged pairs}|$ are non-negative. Putting in the above values, we get three inequalities:

$$\begin{array}{lccc} & i \geq 0 & r - 2i \geq 0 & \frac{n}{2} - r + i \geq 0 \\ \text{Thus} & i \geq 0 & i \leq \frac{r}{2} & i \geq r - \frac{n}{2}. \end{array}$$

These can be combined into the range $i \in [\max(0, r - n/2) \ldots \lfloor r/2 \rfloor]$. These are all the values of $i$ that can give an $(r,i) - subset$. Since any subset $\mathcal{R}$, $|\mathcal{R}| = r$,

is some $(r,i)-subset$ (just count the number of sibling revoked pairs of leaves to find the $i$), the values of $i$ partition all such subsets:

$$\{\mathcal{R} : \mathcal{R} \subseteq \mathcal{N},\ |\mathcal{R}| = r\} = \bigcup_{i \in [\max(0, r-n/2) \ldots \lfloor r/2 \rfloor]} R_{(r,i)}(T). \tag{4.5}$$

We know that $\{\mathcal{R} : \mathcal{R} \subseteq \mathcal{N},\ |\mathcal{R}| = r\} = \binom{n}{r}$, so combining this with Equation (4.4), we get:

$$\sum_{i=\max(0,r-n/2)}^{\lfloor r/2 \rfloor} \binom{n/2}{(i)\ (r-2i)\ (n/2-r+i)} 2^{r-2i} = \binom{n}{r}. \tag{4.6}$$

In summing over $i$ instead of $\mathcal{R}$, we lose a lot of information about the subsets. It is not possible to calculate the size of the cover of an $(r,i)-subset$ from just $r$ and $i$ alone. However, it is possible to obtain some useful information about the size of the cover:

*Lemma* 28. Let $CSRS = (\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme on the binary tree $T$, with $|\mathcal{N}| = n$ users. Let $CSRS' = (\mathcal{N}', \Omega', \gamma')$ be a Complete Subtree Revocation Scheme on the binary tree $T'$, with $|\mathcal{N}'| = n/2$ users, where $T'$ is a subtree of $T$ got by removing the $n$ leaves and the connected edges. Let $\mathcal{R} \subseteq \mathcal{N}$ be a subset with $|\mathcal{R}| = r$ and $\mathcal{R} \in R_{(r,i)}(T)$. Then:

$$t^{CSRS}(\mathcal{N}, \mathcal{R}) = r - 2i + t^{CSRS'}(\mathcal{N}', \mathcal{R}'), \tag{4.7}$$

where $|\mathcal{R}'| = r - i$.

*Proof.* As $\mathcal{R}$ is an $(r,i)-subset$, we know there are $i$ pairs of revoked users, $n/2 - r + i$ pairs of privileged users and $r - 2i$ pairs of mixed users. The mixed pair have specific properties in a Complete Subtree Revocation Scheme independent of the rest of the tree. As one of the leaves is revoked, that leaf, as well as its parent, is in $ST(\mathcal{R})$ ($ST(\mathcal{R})$ is the union of paths from all revoked users to the root). The sibling leaf is privileged, but has a revoked parent. By definition, this node is hanging off, and will require its own index in the cover (i.e. an establishment key in the broadcast). Once these indices have been added to the cover, we can cover the rest of the privileged users in the smaller scheme $CSRS'$. All privileged users in $T$ that were in one of the $r-2i$ mixed pairs are covered, and so the node one level up (which are leaves in $T'$)

can be considered revoked. The pairs of revoked users in $T$ stay revoked in $T'$ (although there will only be half the number of leaves when you go up a level). The only privileged leaves in $T'$ will be the leaves that are parents of pairs of privileged users in $T$. As there are only $n/2 - (r - i)$ such leaves, we have:

$$t^{CSRS}(\mathcal{N}, \mathcal{R}) = r - 2i + t^{CSRS'}(\mathcal{N}', \mathcal{R}'),$$

where $\mathcal{R}'$ is the set of leaves in $T'$ corresponding to nodes in $T$ that are parents of either mixed or revoked pairs of leaves ($|\mathcal{R}'| = r - i$). □

It is this expression of $t(\mathcal{N}, \mathcal{R})$ in terms of the size of a cover in a scheme with half the number of users that forms the basis of the recurrence relation for $t_{aver}(n, r)$. To derive this relation we combine Equation (4.7) with the partition in Formula (4.5).

*Theorem* 29. Let $CSRS = (\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme with $|\mathcal{N}| = n$ users. Let $r$ be an integer, $0 \leq r \leq n$. Then $CSRS$ has:

$$t_{aver}(n, r) = \sum_{i=\max(0, r-n/2)}^{\lfloor r/2 \rfloor} \frac{\binom{n/2}{(i)\ (r-2i)\ (n/2-r+i)} 2^{r-2i}(r - 2i + t_{aver}(n/2, r - i))}{\binom{n}{r}}. \tag{4.8}$$

*Proof.* For any subset $\mathcal{R} \subseteq \mathcal{N}$ of $|\mathcal{R}| = r$ revoked leaves, let $i$ be the number of pairs of revoked users that have the same parent. $\mathcal{R}$ is an $(r, i) - subset$, and by Formula (4.7), we have:

$$t(\mathcal{N}, \mathcal{R}) = r - 2i + t(\mathcal{N}', \mathcal{R}'),$$

where $|\mathcal{N}'| = n/2$ and $|\mathcal{R}'| = r - i$. But we can also go the other way around. We start of with a set $\mathcal{R}' \subseteq \mathcal{N}'$, $|\mathcal{R}'| = r - i$ (same $\mathcal{N}'$ as above). Choose any $i$ of these to be extended to a pair of revoked leaves and the other $r - 2i$ to be extended to a mixed pair. The privileged $n/2 - r + i$ are extended to a pair of privileged leaves. This gives us an $(r, i)$-subset of the leaves $\mathcal{N}$. Therefore, we can enumerate all $(r, i)$-subsets by first considering all subsets of the type $\mathcal{R}'$, and for each of these choosing $i$ leaves to be extended to revoked pairs (choosing $\mathcal{I}$ from $\mathcal{R}'$, $|\mathcal{I}| = i$) . This is sufficient to determine all three types

of pairs. By summing $t(\mathcal{N},\mathcal{R})$ over $\mathcal{R}'$ and $\mathcal{I}'$ instead of $\mathcal{R}$, we get:

$$\sum_{\mathcal{R}\in R_{(r,i)}(T)} t(\mathcal{N},\mathcal{R}) = \sum_{\substack{\mathcal{R}'\subseteq\mathcal{N}'\\|\mathcal{R}'|=r-i}} \sum_{\substack{\mathcal{I}\subseteq\mathcal{R}'\\|\mathcal{I}|=i}} (r-2i+t(\mathcal{N}',\mathcal{R}'))2^{r-2i},$$

where $\mathcal{N}'$ is the set of $n/2$ leaves and $\mathcal{I}$ is the set of those nodes that are extended to revoked pairs (all remaining nodes in $\mathcal{R}'$ are mixed). The power of 2 term is needed because each mixed pair can be in one of two orientations. Swapping a mixed pair has no effect on the size of the cover. Similarly, the choice of $\mathcal{I}$ has no effect on the size of the cover (the leaves in $\mathcal{R}'$ are not changed at all). So we can replace this sum with a binomial coefficient. Also, since $r$ and $i$ are fixed, we can do some rearranging:

$$\begin{aligned}
\sum_{\mathcal{R}\in R_{(r,i)}(T)} t(\mathcal{N},\mathcal{R}) &= \sum_{\substack{\mathcal{R}'\subseteq\mathcal{N}'\\|\mathcal{R}'|=r-i}} \binom{r-i}{i}(r-2i+t(\mathcal{N}',\mathcal{R}'))2^{r-2i}\\
&= \binom{n/2}{r-i}\binom{r-i}{i}2^{r-2i}(r-2i) + \binom{r-i}{i}2^{r-2i}\sum_{\substack{\mathcal{R}'\subseteq\mathcal{N}'\\|\mathcal{R}'|=r-i}} t(\mathcal{N}',\mathcal{R}')\\
&= \binom{n/2}{r-i}\binom{r-i}{i}2^{r-2i}(r-2i) + \binom{r-i}{i}2^{r-2i}\binom{n/2}{r-i}t_{aver}(n/2,r-i)\\
&= \binom{n/2}{r-i}\binom{r-i}{i}2^{r-2i}(r-2i+t_{aver}(n/2,r-i)).
\end{aligned}$$

The product of the two binomial coefficients are just another way of writing the multinomial coefficient $\binom{n/2}{(i)\ (r-2i)\ (n/2-r+i)}$ (choosing $a$ and $b$ from $m$ is the same as choosing $a+b$ from $m$ and then choosing $a$ from $a+b$). The definition for $t_{aver}(n,r)$ is a sum over all $\mathcal{R}\subseteq\mathcal{N}$ of size $r$. By replacing this with the partition of Formula (4.5), we get:

$$\begin{aligned}
t_{aver}(n,r) &= \sum_{\substack{\mathcal{R}\subseteq\mathcal{N}\\|\mathcal{R}|=r}} \frac{t(\mathcal{N},\mathcal{R})}{\binom{n}{r}}\\
&= \frac{\sum_{i=\max(0,r-n/2)}^{\lfloor r/2\rfloor}\sum_{\mathcal{R}\in\{(r,i)-subsets\}} t(\mathcal{N},\mathcal{R})}{\binom{n}{r}}\\
&= \sum_{i=\max(0,r-n/2)}^{\lfloor r/2\rfloor} \frac{\binom{n/2}{(i)\ (r-2i)\ (n/2-r+i)}2^{r-2i}(r-2i+t_{aver}(n/2,r-i))}{\binom{n}{r}}.
\end{aligned}$$

□

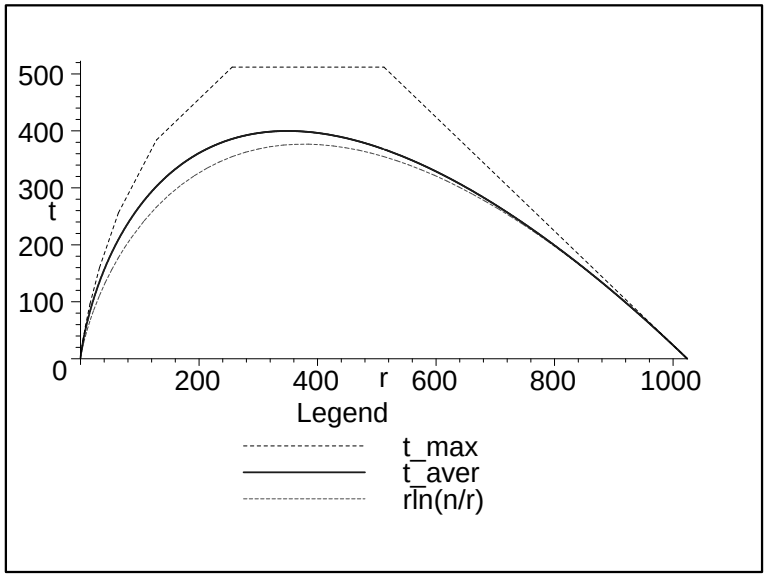


Figure 4.6: $t_{\max}(n, r)$ and $t_{aver}(n, r)$ for the Complete Subtree Revocation Scheme with $n = 2^{10}$. Also plotted is $r \ln(n/r)$.

The entire range of values of $t_{aver}(1024, r)$ is plotted in Figure 4.6, along with $t_{\max}(1024, r)$ for comparison. Calculating the values of $t_{aver}(1024, r)$ explicitly, using Formula (2.2), would be practically impossible. It would require calculating the size of the cover for $2^{1024}$ different subsets. While the shape of $t_{aver}(n, r)$ is broadly similar to that of $t_{\max}(n, r)$, there are several significant differences. Whereas $t_{\max}(n, r)$ was a series of straight lines, $t_{aver}(n, r)$ is a smooth curve from $r = 0$ to $r = n$. The numerical difference between the two varies as well. For the very small and very large values of $r$, the two graphs are only slightly different. But for all values in between, there is a considerable gap.

The recurrence relation, Formula (4.8), allows us to calculate $t_{aver}(n, r)$ much more efficiently working out every cover for the $\binom{n}{r}$ possible subsets, although we do have to work out $t_{aver}(n', r')$, where $n' = n/2$ and $r'$ takes on a range of values. At the first level of recurrence, $t_{aver}(n, r)$ is calculated from $t_{aver}(n/2, r - i)$, for $i$ in the range $[\max(0, r - n/2), \ldots, \lfloor r/2 \rfloor]$. Substituting $r'$ for $r - i$ gives us $t_{aver}(n/2, r')$ for $r'$ in the range $[\lceil r/2 \rceil, \ldots, \min(r, n/2)]$ (as $r - \lfloor r/2 \rfloor = \lceil r/2 \rceil$ and $r - \max(0, r - n/2) = \min(r, n/2)$). Each value

of $t_{aver}(n/2, r')$ is calculated from a range of values of $t_{aver}(n/4, r'')$. Fortunately, these ranges overlap, so storing the results will save on the computation of re-calculating them each time. The range of $r''$ is approximately $[\lceil r/4 \rceil, \ldots, \min(r, n/4)]$. By summing the length of the ranges over all the values of $t_{aver}(n', r')$, we can find out the computational complexity of working out $t_{aver}(n, r)$. We make an upper bound of this in the following lemma:

*Lemma* 30. The recursive relation in Formula (4.8) requires less than $3n/4$ evaluations of $t_{aver}(n, r)$.

*Proof.* At the highest level, to calculate $t_{aver}(n, r)$ we need $t_{aver}(n/2, r')$ for $r' \in [\lceil r/2 \rceil, \ldots, \min(r, n/2)]$. We will show that this range can be no longer than $n/4$ in length. Suppose $r \geq n/2$. This implies $\lceil r/2 \rceil \geq n/4$. Since the upper limit of the range is $\min(r, n/2) = n/2$, the range is at most length $n/4$. Similarly, if $r < n/2$, then the range is $[\lceil r/2 \rceil, \ldots, r]$. This length is at most $r/2 < n/4$.

At lower levels, when we are evaluating the recursive function for $n/4$ or lower, we cannot prove a similar bound. However, the recursive function is defined so that $t_{aver}(n', r')$ is only ever called with $0 \leq r' \leq n'$ ($t_{aver}(n', r')$ is not defined outside this range). So the range of values of $t_{aver}(n', r')$ that can ever be evaluated is of length $n'$ at most. We have already bounded the range for $t_{aver}(n/2, r')$, so that just leaves the powers of two from $n/4$ to 2. This means that the total number of evaluations of $t_{aver}(n', r')$ needed is bounded by:

$$\frac{n}{4} + \sum_{i=2}^{\log_2(n)} \frac{n}{2^i} \quad < \quad \frac{n}{4} + n \sum_{i=2}^{\infty} \frac{1}{2^i} \quad = \quad \frac{n}{4} + \frac{n}{2} \quad = \quad \frac{3n}{4}. \qquad \square$$

A more detailed analysis would show that the number of operations is in fact bounded by $n/2$. The important point, however, is that it costs $\mathcal{O}(n)$ operations, compared to $\mathcal{O}(\binom{n}{r}) = \mathcal{O}(\min(n^r, n^{n-r}))$ operations needed to calculate the average explicitly.

We have also plotted $r \ln(n/r)$ in Figure 4.6. It is believed that this serves as a lower bound for $t_{aver}(n, r)$, and it is true for all values of $n, r$ up to $2^{10}$.

*Conjecture* 31. Let $\mathcal{N}$ be a set of $n$ users. Let $CSRS = (\mathcal{N}, \Omega, \gamma)$ be a

Complete Subtree Revocation Scheme with $|\mathcal{N}| = n$ users. Then $CSRS$ has:

$$t_{aver}(n, r) \geq r \ln\left(\frac{n}{r}\right).$$

This has yet to be proven.

A final note on the Complete Subtree Revocation Scheme. If we look at the subtrees of a complete binary tree rooted at the two children of the root, then we get two complete binary trees of depth 1 less. Since all the nodes in a complete tree represent all the indices in the scheme, a Complete Subtree Revocation Scheme on $n$ users is comprised of two copies of the same scheme on two (different) sets of $n/2$ users, plus a key for all $n$ users. This is a property common to many tree based schemes, and is something we will exploit in Chapter 5.

## 4.3 Complete Subtree with a-ary tree

As mentioned in Chapter 3, there is a generalisation of the Complete Subtree Revocation Scheme to an $a$-ary tree, for any integer $a \geq 3$. There are in fact two generalisations, but we will ignore the simple one as it significantly increased the bandwidth and only slightly reduces storage (compared to the Complete Subtree on a binary tree). Instead we will focus on the combination of the Complete Subtree and the Power Set Method as described in [1] and Section 3.1.

While the structure of an $a$-ary tree is not very different from that of a binary tree, some of the concepts we have been using need to be re-evaluated. We can no longer identify the children of a node as being either left or right, they must be numbered $1^{st}$ child, $2^{nd}$ child up to $a^{th}$ child. This also means that the sibling of a node is not well defined. Any node (except the root) has $a - 1 \geq 2$ siblings. As before we can define a Steiner Tree $ST(\mathcal{R})$ on a set of leaves $\mathcal{R}$. In the binary tree any internal node that was in $ST(\mathcal{R})$ either has one child in $ST(\mathcal{R})$ (and so has a node hanging off), or has both children in $ST(\mathcal{R})$. There are more possibilities in the $a$-ary tree as it can have anywhere from 1 to $a$ children in $ST(\mathcal{R})$. As a result, more than one node can hang off the same node in an $a$-ary tree.

### 4.3.1 Maximum Bandwidth

The formula for $t_{\max}(n, r)$, and the associated proof, is similar to that of the binary tree. The most important difference stems from how $t(\mathcal{N}, \mathcal{R})$ is calculated. The subtree $ST(\mathcal{R})$ is constructed, and the nodes that hang off are the ancestors of all privileged users/leaves. With the Complete Subtree on a binary tree, it was simply a matter of one index for every node hanging off $ST(\mathcal{R})$. Each index would cover all leaves descended from one node in the tree. When we use an $a$-ary tree, the indices are more complex. Each node $v_i$ on the tree has several indices, one for every non-empty proper subset of children, identified by the bit string $B$. Any particular index $(i, B)$ will cover the leaves descended from those children of $v_i$ that appear as a 1 in the bit string $B$. In order to cover $\mathcal{N} \setminus \mathcal{R}$, we look at each node in $ST(\mathcal{R})$. If any node has all its children in $ST(\mathcal{R})$, then we do not need any index for that node. If any node, $v_i$, has at least one child not in $ST(\mathcal{R})$, then we add the index $(i, B)$ to the cover where the $i^{th}$ bit of $B$ is 1 if and only if the $i^{th}$ child of $v_i$ is not in $ST(\mathcal{R})$. Each node in $ST(\mathcal{R})$ must have at least one child in $ST(\mathcal{R})$ as the subtree is just the union of paths from leaves to the root. Since we will never have an internal node in $ST(\mathcal{R})$ with all children not in $ST(\mathcal{R})$, the index $(i, 11 \ldots 1)$ does not appear in this scheme for any internal node $v_i$. However, if $r = 0$ then the root will have all children not in $ST(\mathcal{R})$, which is why the index $(root, 11 \ldots 1)$ is added to the scheme. Whereas with the binary tree, the size of the cover was the number of nodes that hung off $ST(\mathcal{R})$, with an $a$-ary tree, it is the number of nodes in $ST(\mathcal{R})$ with at least one node hanging off.

In the following lemma, we will present three conditions on $\mathcal{R}$ that gives $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. We will then show that these specify $\mathcal{R}$ enough to calculate $t(\mathcal{N}, \mathcal{R})$. The techniques used in Lemma 32 are the same as those used in Lemma 24 and Theorem 25. We assume that the condition on $\mathcal{R}$ does not hold, and then show a subset $\mathcal{R}'$ with a larger cover can be constructed.

*Lemma* 32. Let $CSRS_a$ be a Complete Subtree Revocation Scheme on an $a$-ary tree. Let $n = a^k$, and $j = \lfloor \log_a(r) \rfloor$. If $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ then the following must be true of $\mathcal{R}$ (and $ST(\mathcal{R})$):

1. All nodes at depth $j-1$ or lower (distance $j-1$ to the root or less) have all children in $ST(\mathcal{R})$

2. Nodes at depth $j+1$ or greater are either not in $ST(\mathcal{R})$ or only have 1 child in $ST(\mathcal{R})$

3. A node at depth $j$ will only have all children in $ST(\mathcal{R})$ if all other nodes at lower depth have either $a$ or $a-1$ children in $ST(\mathcal{R})$

*Proof.* Proof of *1.* Let $\mathcal{R} \subseteq \mathcal{N}$ be a subset with $|\mathcal{R}| = r$ and $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. The number of nodes in any level of an complete $a$-ary tree is $a$ to the power of the distance to the root. Therefore, there are $a^j$ nodes in the tree at depth $j$. We also know that $a^j \leq r$ since $j = \lfloor \log_a(r) \rfloor$. So in the complete tree, there must be less than or equal to $r$ nodes at depth $j$ that are in $ST(\mathcal{R})$. Suppose the first condition does not hold, that there is at least one node (say $v_{i_1}$) at depth $\leq j-1$ that has at least one child not in $ST(\mathcal{R})$. There must be at least one node at depth $j$ not in $ST(\mathcal{R})$. So the number of nodes at depth $j$ that are in $ST(\mathcal{R})$ is strictly less than $r$.

Every node with 2 or more children in $ST(\mathcal{R})$ increases the number of nodes in $ST(\mathcal{R})$ in the next level. Since we have less than $r$ nodes in $ST(\mathcal{R})$ at depth $j$ and must have exactly $r$ at the level of the leaves, there must be at least one such node between the nodes at depth $j$ and the level above the leaves. Let $v_{i_2}$ be the lowest such node in $ST(\mathcal{R})$. As $v_{i_2}$ is the lowest node with more than one child in $ST(\mathcal{R})$, each of these children of $v_{i_2}$ has only has one descendant leaf in $ST(\mathcal{R})$. Suppose we define $\mathcal{R}'$ to be $\mathcal{R}$ without one of these leaves. The cover of $\mathcal{N} \setminus \mathcal{R}'$ would still require an index for node $v_{i_2}$. The node $v_{i_2}$ is still in $ST(\mathcal{R}')$ (it had more than one child in $ST(\mathcal{R})$) and has at least one child not in $ST(\mathcal{R}')$ (the child on the path we removed). All indices for nodes lower down this path would be gone from the cover. Since the highest $v_{i_2}$ could be is $k-j$, there are $k-j-1$ less nodes in the cover. In order to have $|\mathcal{R}'| = r$, we need to also add a leaf to $\mathcal{R}'$. Suppose we add one of the leaves descended from $v_{i_1}$. As $v_{i_1}$ is at depth $\leq j-1$, there are at least $k-j$ nodes that have at least one node hanging off this path. Therefore $t(\mathcal{N}, \mathcal{R}') > t(\mathcal{N}, \mathcal{R})$, which contradicts the assumption on $\mathcal{R}$.

Proof of *2.* Since $a^j$ is the greatest power of $a$ less than or equal to $r$, we

have that $a^{j+1} > r$. So at least one node at depth $j$ has a child not in $ST(\mathcal{R})$. If any node in depth $\geq j+1$ has more than one child in $ST(\mathcal{R})$, we would have the same contradiction as above. We cannot have a node hanging off $ST(\mathcal{R})$ when there is a node lower down the tree with two or more children in $ST(\mathcal{R})$. Since we must have a node hanging off at least one node at depth $j$, all nodes at depth $j+1$ and lower that are in $ST(\mathcal{R})$ have exactly one child in $ST(\mathcal{R})$.

Proof of *3.* Suppose the third condition does not hold. We have one node, $v_{i_1}$, at depth $j$ with all children in $ST(\mathcal{R})$ and another node, $v_{i_2}$, at the same depth with two or more children not in $ST(\mathcal{R})$. We know there are no nodes with more than one child in $ST(\mathcal{R})$ at depth $\geq j+1$, so each node at depth $j+1$ that is in $ST(\mathcal{R})$ will only have one descendant leaf in $ST(\mathcal{R})$. Define $\mathcal{R}'$ to be $\mathcal{R}$ but with one of the leaves descended from $v_{i_1}$ descended from one of the children $v_{i_2}$ that is not in $ST(\mathcal{R})$. In $ST(\mathcal{R})$, $v_{i_1}$ had all children in $ST(\mathcal{R})$, and so did not require an index for the cover. Because one of its descendant leaves is not present in $ST(\mathcal{R}')$, it now has a child not in $ST(\mathcal{R}')$ and so must have a index for the cover. Even though we had to add a leaf to $ST(\mathcal{R}')$ descended from $v_{i_2}$, since it had two or more children not in $ST(\mathcal{R})$, it will still have at least one child not in $ST(\mathcal{R})'$ and so still require an index. So we get that $t(\mathcal{N}, \mathcal{R}') > t(\mathcal{N}, \mathcal{R})$, which contradicts the assumption on $\mathcal{R}$. □

The first and second conditions on $ST(\mathcal{R})$ are the same for the Complete Subtree Revocation Scheme on a binary tree. $ST(\mathcal{R})$ has the appearance of a complete tree at the top and individual paths at the bottom. The third condition is unique to an $a$-ary tree, however, and would be redundant in a binary tree. The first condition means all nodes at depth $j$ are in $ST(\mathcal{R})$ and so automatically have either 1 or 2 ($a-1$ or $a$ if $a=2$) children in $ST(\mathcal{R})$. In the case where $a > 2$, this condition will give us two difference cases for the structure of $ST(\mathcal{R})$. This will lead to two cases in the formula for $t_{\max}(n, r)$.

*Corollary* 33. Let $CSRS_a$ be a Complete Subtree Revocation Scheme on an $a$-ary tree. Let $n = a^k$, and $j = \lfloor \log_a(r) \rfloor$. Then $CSRS_a$ has:

$$t_{\max}(n, r) = \begin{cases} r(k-j-1) + a^j & \text{if } r \leq (a-1)a^j \\ r(k-j-2) + a^{j+1} & \text{otherwise} \end{cases}. \tag{4.9}$$

*Proof.* Since $j = \lfloor \log_a(r) \rfloor$, we know that $r$ must lie in the range $a^j \leq r < a^{j+1}$. We first consider the case when $a^j \leq r \leq (a-1)a^j$. We know from the first condition in Lemma 32 that all nodes at depth $j$ are in $ST(\mathcal{R})$, and consequently each node must have at least one child in $ST(\mathcal{R})$ (paths only stop at the leaves). As a result of the second condition there are $r$ nodes at depth $j+1$ in $ST(\mathcal{R})$. No node further down the tree has more than one child in $ST(\mathcal{R})$ and that is the only way that the number of nodes in $ST(\mathcal{R})$ can increase from level to level.

Suppose there was a node at depth $j$ with all $a$ children in $ST(\mathcal{R})$. By the third condition in Lemma 32, all other nodes at that level must have at least $a-1$ children in $ST(\mathcal{R})$. That gives a total of $a + (a^j - 1)(a-1)$ nodes at depth $j+1$ in $ST(\mathcal{R})$:

$$\begin{aligned} a + (a^j - 1)(a-1) &= 1 + (a-1) + (a^j - 1)(a-1) \\ &= 1 + a^j(a-1). \end{aligned}$$

But, $r \leq (a-1)a^j < 1 + a^j(a-1)$, which means that there are more than $r$ nodes at depth $j+1$ in $ST(\mathcal{R})$. This contradicts the fact that $ST(\mathcal{R})$ has $r$ leaves. Therefore, for $r$ in the above range, there are no nodes at depth $j$ with all $a$ children in $ST(\mathcal{R})$. Each node at this level will then require one index for the cover, and there are $a^j$ of them. From depth $j+1$ down, $ST(\mathcal{R})$ is just $r$ separate paths of length $k-j-1$. As each node in each path has only one child in $ST(\mathcal{R})$ (or more importantly, at least one node hanging off), they will all require indices for the cover. Therefore:

$$t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r) = r(k-j-1) + a^j.$$

Conversely, if $r > (a-1)a^j$ then there must be some nodes at depth $j$ with all children in $ST(\mathcal{R})$. Exactly how many there are depends on the difference between $r$ and $(a-1)a^j$. We can only have such a node if all other nodes have at least $a-1$ children in $ST(\mathcal{R})$. So we know there are no less than $a-1$ children in $ST(\mathcal{R})$ for each node, meaning at least $(a-1)a^j < r$ nodes in $ST(\mathcal{R})$ at depth $j+1$. If each node at depth $j$ had exactly $a-1$ children in $ST(\mathcal{R})$, then there would be $(a-1)a^j$ nodes at depth $j+1$ in $ST(\mathcal{R})$. In order to have $r$ nodes in $ST(\mathcal{R})$ at depth $j+1$, we need exactly $r - (a-1)a^j$ nodes

at depth $j$ to have all $a$ children in $ST(\mathcal{R})$. The remaining $a^j - (r - (a-1)a^j)$ nodes at this level will have one child not in $ST(\mathcal{R})$ hanging off. From depth $j+1$, $ST(\mathcal{R})$ is the same as when $r \leq (a-1)a^j$, $r$ paths of length $k-j-1$. Therefore:

$$\begin{aligned} t(\mathcal{N},\mathcal{R}) = t_{\max}(n,r) &= r(k-j-1) + a^j - (r - (a-1)a^j) \\ &= r(k-j-1) + a^j - r + a^{j+1} - a^j \\ &= r(k-j-2) + a^{j+1}. \end{aligned}$$ □

The formula we found for the binary case was $t_{\max}(n,r) = r(k-j) - 2(r-2^j)$ (Formula (4.1)). If we were to put $a = 2$ into Formula (4.9) we would get the same. The condition that $r \leq (a-1)a^j = 2^j$ can only be true when $r = 2^j$ as by the definition of $j$, $r \geq 2^j$. If $r = 2^j$, then Formula (4.9) gives:

$$\begin{aligned} t_{\max}(n,r) = r(k-j-1) + a^j &= r(k-j-1) + r \\ &= r(k-j) \\ &= r(k-j) - 2(r-2^j), \end{aligned}$$

since $r - 2^j = 0$. For any other value of $r$ we have:

$$\begin{aligned} t_{\max}(n,r) = r(k-j-2) + a^{j+1} &= r(k-j) - 2r + 2^{j+1} \\ &= r(k-j) - 2(r-2^j). \end{aligned}$$

Figure 4.7 shows $t_{\max}(n,r)$ for a binary, a ternary and a quaternary tree. The binary tree has height 10, $n = 2^{10} = 1024$, the ternary tree height 6, $n = 3^6 = 729$ and the quaternary tree height 5, $n = 4^5 = 1024$. If we want to use $t_{\max}(n,r)$ to rate the performance the schemes, then the fact that the ternary tree based scheme has a smaller user set must be taken into account when comparing the plots. From simple appearances, the ternary tree scheme seems to perform the best. Its plot for $t_{\max}(n,r)$ goes no higher than 243. The quaternary tree scheme levels off at 256, just a little higher. But these plateaus are actually $n/3$ and $n/4$ respectively (in accordance with Formula (3.3)). Relative to the population size, the quaternary tree scheme performs better than the ternary tree scheme. Both perform better than the binary tree scheme. This is obvious for the quaternary tree scheme as they

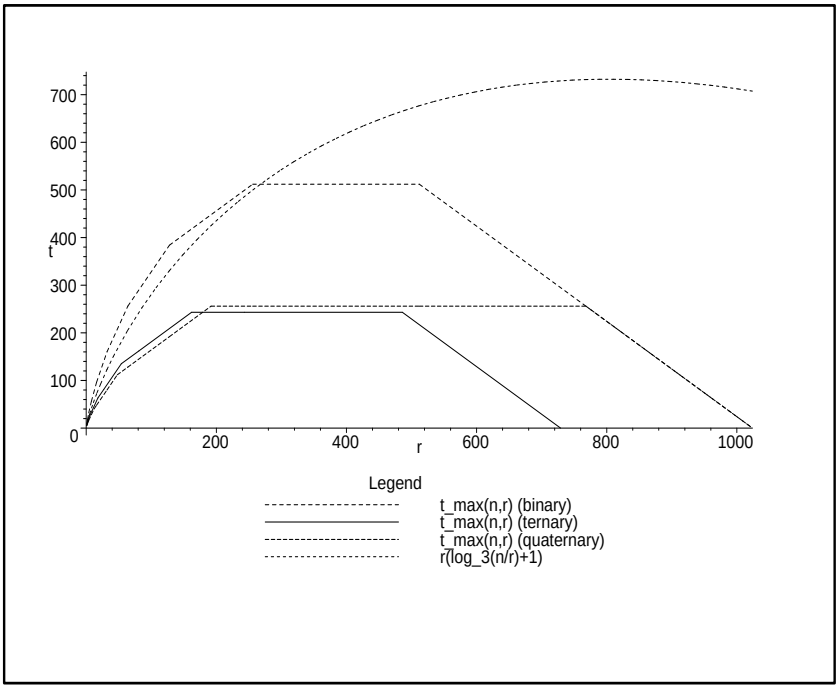


Figure 4.7: $t_{\max}(n, r)$ for the Complete Subtree on a binary, ternary and quaternary tree.

have the same population and so can be compared directly. And we know that the plateau for the binary tree scheme is $n/2$ which is worse than $n/3$ for the ternary tree scheme. We shall be looking at how to compare different schemes on different user sets in Chapter 7. The figure also shows the original upper bound of Asano of $t_{\max}(n, r) \leq r(\log_a(n/r) + 1)$ in the instance of the ternary tree when $n = 3^6 = 729$. The bound quickly diverges from the true maximum and is only close to $t_{\max}(n, r)$ for small values of $r$.

### 4.3.2 Average Bandwidth

In the Section 4.2 we derived a recursive formula for $t_{aver}(n, r)$ for the Complete Subtree Revocation Scheme on a binary tree. We did this by expressing the cover in the binary tree with $n$ leaves in terms of the binary tree with one less level, i.e. $n/2$ leaves. Any revoked subset $\mathcal{R} \subseteq \mathcal{N}$ would place each parent of a pair of leaves into one of three types: both revoked, one revoked, or neither revoked. In terms of the cover it does not matter which leaf is revoked. The same approach works for an $a$-ary tree. There are $2^a$ possible variations of $a$

leaves descended from the same parent if each leaf can be either privileged or revoked. To be able to calculate the size of the cover we only need consider the three types: all revoked, some revoked and none revoked.

In the method described to find the cover of $\mathcal{N} \setminus \mathcal{R}$, an index for an internal node in the $a$-ary tree will occur in the cover if that node is in $ST(\mathcal{R})$ and has at least one child not in $ST(\mathcal{R})$. So any node that is a parent of $a$ revoked leaves will not require an index for the cover. At the other extreme, a parent of $a$ privileged leaves will not have an index in the cover either. The scheme only has indices for proper subsets of privileged children of any node, with the sole exception of the root. If all children of node $v$ are privileged, then we use the index $(par(v), B)$ (or possibly an ancestor of $v$ even higher up the tree), where $B$ corresponds to the subset of children of $par(v)$ including only $v$ and any other sibling that also has only privileged children. The only nodes at the level above the leaves whose indices appear in the cover are those of the third category, those with at least one but not all leaves revoked. For each such node, we add to the cover the index of that node and the bit-string corresponding to the subset of the privileged children. As all privileged children are covered, the parent can be considered revoked when it comes to covering the rest of the tree. So for any subset $\mathcal{R} \subseteq \mathcal{N}$, if we define $Ar, Sr, Nr$ to be the sizes of the sets of nodes at the level above the leaves with all, some and no revoked leaves respectively, we have:

$$t(\mathcal{N}, \mathcal{R}) = Sr + t(\mathcal{N}', \mathcal{R}'),$$

where $\mathcal{N}'$ is the set of $n/a$ parents of leaves and $\mathcal{R}'$ is the set of nodes whose children are either all revoked or have some leaves revoked: $|\mathcal{R}'| = Ar + Sr$.

In order to derive a recursive relation on $t_{aver}(n, r)$, we need to be able to use the parameters $Ar, Sr, Nr$ to sum over all $r$ subsets from $n$. There are only two degrees of freedom for the parameters as $Ar + Sr + Nr = n/a$. We need to know how many $(Ar, Sr, Nr)$ triples can give an $|\mathcal{R}| = r$ subset, and how many such subsets there are for each triple. The first part is simply a matter of placing some bounds on $Nr$ and $Sr$ (we will fix $Ar = n/a - Nr - Sr$).

*Lemma* 34. Let $T$ be a complete $a$-ary tree with $n = a^k$ leaves. Let $\mathcal{R}$ be a non-empty subset of leaves of $T$ with $|\mathcal{R}| = r \geq 1$. Let $Ar$, $Sr$ and $Nr$ be the

number of nodes in the level directly above the leaves with all, some (between 1 and $a-1$) and no revoked leaves as children, respectively. Then:

$$Ar = \frac{n}{a} - Nr - Sr \tag{4.10}$$

$$Nr \in \left[\max\left(0, \frac{n}{a} - r\right) \ldots \frac{n-r}{a}\right] \tag{4.11}$$

$$Sr \in \left[\max\left(0, \frac{n-r-a.Nr}{a-1}\right) \ldots \min\left(\frac{n}{a} - Nr, n-r-a.Nr\right)\right] \tag{4.12}$$

*Proof.* We know that $(Ar, Sr, Nr)$ all have to be non-negative as well as bounded above by $n/a$. Since there are $n/a$ nodes in the level directly above the leaves, we have:

$$\begin{aligned} \frac{n}{a} &= Ar + Sr + Nr \\ \text{Hence} \quad Ar &= \frac{n}{a} - Nr - Sr. \end{aligned} \tag{4.13}$$

We will consider the ranges of each parameter separately, starting with $Nr$.

$Nr$ is bounded above by $(n-r)/a$, as the number of privileged leaves descended from the $Nr$ nodes, $a.Nr$, has to be less than or equal $n-r$ (the total number of privileged leaves in $T$). The very minimum value $Nr$ can take would be when $Ar + Sr$ takes the maximum value. Since the $Ar + Sr$ nodes either have all or some children revoked, they must each have at least one revoked child. Therefore:

$$\begin{aligned} Ar + Sr &\leq \min\left(\frac{n}{a}, r\right) \\ \text{So that} \qquad Nr &\geq \frac{n}{a} - \min\left(\frac{n}{a}, r\right) \qquad \text{by (4.13)} \\ &= \max\left(0, \frac{n}{a} - r\right) \\ \text{Hence} \qquad Nr &\in \left[\max\left(0, \frac{n}{a} - r\right) \ldots \frac{n-r}{a}\right]. \end{aligned}$$

Now we will find the range for $Sr$. Naturally, $Sr \geq 0$. We know that the sum of the revoked leaves in the $Ar$ subsets and the $Sr$ subsets have to add up to $r$. Each $Ar$ subset will have exactly $a$ revoked leaves, whereas the $Sr$ subsets can have anywhere between 1 and $a-1$ leaves. Therefore $r$ must be in the range:

$$Sr + a.Ar \leq r \leq (a-1)Sr + a.Ar.$$

Substituting in $n/a - Nr - Sr$ for $Ar$, and expressing the bounds in terms of $Sr$, we get:

$$\begin{aligned} Sr + a\left(\frac{n}{a} - Nr - Sr\right) &\leq r & r &\leq (a-1)Sr + a\left(\frac{n}{a} - Nr - Sr\right) \\ -(a-1)Sr + n - a.Nr &\leq r & r &\leq -Sr + n - a.Nr \\ Sr &\geq \frac{(n - r - a.Nr)}{a-1} & Sr &\leq n - r - a.Nr. \end{aligned}$$

Finally, we have bounds on $Ar$ which we need to put in terms of $Sr$ and $Nr$. We know $Ar$ must be positive, and also $Ar \leq r/a$, as there are $a$ revoked leaves for every $Ar$ node. However, the latter bound is a weaker statement than $r \leq (a-1)Sr + a.Ar$, and so is redundant. By substituting in Formula (4.10) into the former bound, we get:

$$\begin{aligned} & & Ar &\geq 0 \\ \text{i.e.} & & \frac{n}{a} - Nr - Sr &\geq 0 \\ \text{So} & & Sr &\leq \frac{n}{a} - Nr. \end{aligned}$$

All this combined gives the following range for $Sr$:

$$Sr \in \left[\max\left(0, \frac{(n - r - a.Nr)}{a-1}\right) \ldots \min\left(\frac{n}{a} - Nr, n - r - a.Nr\right)\right]. \qquad \square$$

Lemma 34 gives us bounds on $Ar$, $Sr$ and $Nr$ from any $\mathcal{R} \subseteq \mathcal{N}$, with $|\mathcal{R}| = r$. However, we need to show that every triple $(Ar, Sr, Nr)$ that satisfies Formulae (4.10), (4.11) and (4.12) corresponds to a $r$-subset in order show that these triples can generate all $r$-subsets.

*Lemma* 35. Let $T$ be a complete $a$-ary tree with $n = a^k$ leaves ($a \geq 2$). For all triples $(Ar, Sr, Nr)$ satisfying Formulae (4.10), (4.11) and (4.12), there exists a subset $\mathcal{R}$ with $|\mathcal{R}| = r$, such that the number of nodes one level above the leaves with all children revoked, some (between 1 and $a-1$) children revoked and no children revoked is $Ar$, $Sr$ and $Nr$ respectively.

*Proof.* Let $(Ar, Sr, Nr)$ be a solution to Formulae (4.10), (4.11) and (4.12). Obviously $Sr$ and $Nr$ are non-negative. And $Sr \leq n/a - Nr$, so:

$$Ar = \frac{n}{a} - Nr - Sr \geq \frac{n}{a} - Nr - \frac{n}{a} + Nr = 0.$$

So all three values $(Ar, Sr, Nr)$ are non-negative. By Formula (4.10), all three sum to $n/a$ (which also means they are each bounded above by $n/a$). So it is possible to choose a subset of the $n$ leaves with $Ar$, $Sr$ and $Nr$ of the nodes one level above the leaves with all, some and no children revoked respectively. It remains to show that we can choose a subset with exactly $r$ leaves.

By substituting $Nr = n/a - Ar - Sr$ into the two bounds on $Sr$:

$$Sr \geq \frac{(n - r - a.Nr)}{a - 1} \qquad Sr \leq n - r - a.Nr,$$

and expressing them in terms of $r$, we get:

$$Sr + a.Ar \leq r \leq (a - 1)Sr + a.Ar,$$

(this is the reverse of the process in Lemma 34). Therefore, there exists $Sr$ integers $i_1, \ldots, i_{Sr}$ with $1 \leq i_j \leq a - 1$, for $j = 1, \ldots, Sr$, such that:

$$a.Ar + \sum_{j=1}^{Sr} i_j = r.$$

Define $\mathcal{R}$ as follows. Choose any $Nr$ nodes at one level above the leaves to have no revoked children. Choose any $Ar$ from the remaining nodes to have all revoked children. Let the number of revoked children of the remaining $Sr$ nodes ($Ar + Sr + Nr = n/a$) be $i_1, \ldots, i_{Sr}$. Since $a.Ar + \sum_{j=1}^{Sr} i_j = r$, $\mathcal{R}$ has $r$ leaves. □

For conciseness, we will use $r_1$ to represent the range in Formula (4.11) and $r_2$ to represent the range in Formula (4.12).

We know the different $(Ar, Sr, Nr)$ triples that give rise to an $\mathcal{R}$ subset of size $r$. But in order to be able to use the triples to sum over all subsets of size $r$, we need to know how many choices there are for a given triple that result in an $r$-subset. It is not as simple as counting the number of solutions to the equation $a.Ar + \sum_{j=1}^{Sr} i_j = r$. For any one solution, we can generate the subset $\mathcal{R}$ by choosing $Ar$ nodes to have all children revoked and $Sr$ nodes to have $i_1, \ldots, i_{Sr}$ revoked children (choosing from the nodes at height 1). So for $j = 1, \ldots, Sr$, there are $\binom{a}{i_j}$ possibilities for the subsets of children of the corresponding node, giving $\prod_{j=1}^{Sr} \binom{a}{i_j}$. We need to work out how many ways there are to generate $\mathcal{R}$ using all possible solutions to $a.Ar + \sum_{j=1}^{Sr} i_j = r$. In order to calculate this factor, we will use the following recursive relation.

*Lemma* 36. Let $ns(x, y)$ be the number of ways of picking the subsets $S_1, \ldots, S_y$, $S_i \subset \{1, \ldots, a\}$ with $1 \leq |S_i| \leq a - 1$ and $\sum_{i=1}^{y} |S_i| = x$. Then:

$$ns(x,y) = \begin{cases} 1 & \text{if } x = y = 0 \\ 0 & \text{if } x \leq y \text{ or } x \geq (a-1)y \text{ , and} \\ \binom{a}{x} & \text{if } 1 \leq x \leq a-1 \\ \displaystyle\sum_{i=\max(1,x-(y-1)(a-1))}^{\min(a-1,x-y+1)} \binom{a}{i} ns(x-i, y-1) & \text{otherwise.} \end{cases}$$

*Proof.* The extreme values of $ns(x, y)$ are trivial to work out. If both $x$ and $y$ are zero, then the sum of $|S_i|$ will equal zero as there are no subsets. This gives exactly 1 way of satisfying the conditions, $ns(0, 0) = 1$. Because of the limitation on the size of each subset, $1 \leq |S_i| \leq a - 1$, the sum of the sizes of the $y$ subsets must be within $y \leq \sum |S_i| \leq (a-1)y$. As this sum has to equal $x$, for any $x$ outside the range, it is not possible to create a set of subsets with the properties we desire. Therefore for any $x \leq y$ or $x \geq (a-1)y$, we have $ns(x, y) = 0$. This includes the two cases $x = 0, y > 0$ and $x > 0, y = 0$. If $y = 1$ we will either have $x \geq a$, in which case we cannot choose 1 subset with $|S_1| = x$ (since $|S_i| \leq a - 1$), or $x \leq a - 1$. In the latter case, $ns(x, 1) = \binom{a}{x}$ as we are choosing a set of size $x$ from $a$.

We have covered all cases where $y = 0$ or 1. We shall use a recursive argument to reduce $y$ for any larger values. We can consider one of the subsets separately from the rest. As the order of the subsets $S_i$ is unimportant, without loss of generality consider the first, $S_1$. We know that $1 \leq |S_1| \leq a - 1$. If this subset is size $i$, then there are $\binom{a}{i}$ possible ways of choosing such a subset. We are also left with counting the number of ways to pick $y - 1$ subsets whose sizes add up to $x - i$, which is just $ns(x - i, y - 1)$. So the total number of ways of picking $y$ subsets that sum to $x$ and where the first subset is size $i$ is $\binom{a}{i} ns(x - i, y - 1)$. We need to sum this term over all possible values of $i$. The bound $1 \leq i \leq a - 1$ may not be tight as it could result in $x - i$ that cannot be made from the sum of $y - 1$ subsets. We must also have that

$y - 1 \leq x - i \leq (y-1)(a-1)$. In terms of $i$, these become:

$$\begin{aligned} i &\geq 1 & i &\leq a-1 \\ i &\geq x-(y-1)(a-1) & i &\leq x-y+1, \end{aligned}$$

and so the formula for $ns(x,y)$, with $y \geq 2$, is:

$$ns(x,y) = \sum_{i=\max(1,x-(y-1)(a-1))}^{\min(a-1,x-y+1)} \binom{a}{i} ns(x-i, y-1). \qquad \square$$

In the proof for the formula for $t_{aver}(n,r)$, we need to express a sum over all subsets $\mathcal{R}$, with $\mathcal{R} \subseteq \mathcal{N}$ and $|\mathcal{R}| = r$ in terms of its compliment.

*Lemma* 37. Let $S_1$ be the following set of subsets: $\{\mathcal{R} : \mathcal{R} \subseteq \mathcal{N}, |\mathcal{R}| = r\}$, where $|\mathcal{N}| = n$. Let $S_2$ be the following set of subsets: $\{\mathcal{N} \setminus \mathcal{R}' : \mathcal{R}' \subseteq \mathcal{N}, |\mathcal{R}'| = n - r\}$. Then $S_1 = S_2$.

*Proof.* Let $\mathcal{R}$ be any subset in $S_1$. Consider the subset $\mathcal{N} \setminus \mathcal{R}$. Obviously, $\mathcal{N} \setminus \mathcal{R} \subseteq \mathcal{N}$. Since $\mathcal{R} \subseteq \mathcal{N}$, we have:

$$\begin{aligned} & \mathcal{R} \cap (\mathcal{N} \setminus \mathcal{R}) = \emptyset & (4.14) \\ \text{Hence} \quad & |\mathcal{R}| + |(\mathcal{N} \setminus \mathcal{R})| = |\mathcal{N}| \\ \text{So} \quad & |(\mathcal{N} \setminus \mathcal{R})| = |\mathcal{N}| - |\mathcal{R}| \\ \text{i.e.} \quad & |(\mathcal{N} \setminus \mathcal{R})| = n - r. \end{aligned}$$

Equation (4.14) also implies that $\mathcal{R} = \mathcal{N} \setminus (\mathcal{N} \setminus \mathcal{R})$. Therefore, $\mathcal{R} \in S_2$, which implies $S_1 \subseteq S_2$.

Let $\mathcal{N} \setminus \mathcal{R}'$ be any subset in $S_2$. Clearly, $\mathcal{N} \setminus \mathcal{R}' \subseteq \mathcal{N}$. Since $\mathcal{R}' \subseteq \mathcal{N}$, we have:

$$\begin{aligned} & \mathcal{R}' \cup (\mathcal{N} \setminus \mathcal{R}') = \emptyset \\ \text{Hence} \quad & |\mathcal{R}'| + |(\mathcal{N} \setminus \mathcal{R}')| = |\mathcal{N}| \\ \text{So} \quad & |(\mathcal{N} \setminus \mathcal{R}')| = |\mathcal{N}'| - |\mathcal{R}'| \\ \text{i.e} \quad & |(\mathcal{N} \setminus \mathcal{R}')| = n - (n-r) = r. \end{aligned}$$

Therefore $\mathcal{N} \setminus \mathcal{R}' \in S_1$, so $S_2 \subseteq S_1$. Coupled with the above, this means $S_1 = S_2$. $\square$

*Corollary* 38. Let $RS = (\mathcal{N}, \Omega, \gamma)$ be any Revocation Scheme on $n$ users. For any $r \leq n$, $RS$ has:

$$\sum_{\substack{\mathcal{R}\subseteq\mathcal{N} \\ |\mathcal{R}|=r}} \frac{t(\mathcal{N}, \mathcal{N}\setminus\mathcal{R})}{\binom{n}{r}} = t_{aver}(n, n-r).$$

*Proof.* The result comes directly from Lemma 37:

$$\begin{aligned}
\sum_{\substack{\mathcal{R}\subseteq\mathcal{N} \\ |\mathcal{R}|=r}} \frac{t(\mathcal{N}, \mathcal{N}\setminus\mathcal{R})}{\binom{n}{r}} &= \sum_{\substack{\mathcal{R}\subseteq\mathcal{N} \\ |\mathcal{R}|=n-r}} \frac{t(\mathcal{N}, \mathcal{R})}{\binom{n}{r}} \quad \text{by Lemma 37} \\
&= \sum_{\substack{\mathcal{R}\subseteq\mathcal{N} \\ |\mathcal{R}|=n-r}} \frac{t(\mathcal{N}, \mathcal{R})}{\binom{n}{n-r}} \quad \text{since } \binom{n}{r} = \binom{n}{n-r} \\
&= t_{aver}(n, n-r). \qquad \square
\end{aligned}$$

We can now prove the formula for $t_{aver}(n, r)$.

*Theorem* 39. Let $CSRS_a$ be a Complete Subtree Revocation Scheme on an $a$-ary tree with $n = a^k$ users. Then for $r \geq 1$, $CSRS_a$ has:

$$t_{aver}(n, r) = \frac{\sum_{\substack{Nr\in r_1 \\ Sr\in r_2}} \binom{n/a}{Nr\ Sr\ Ar} ns(r - a.Ar, Sr)\left(Sr + t_{aver}\left(\frac{n}{a}, \frac{n}{a} - Nr\right)\right)}{\binom{n}{r}}. \tag{4.15}$$

*Proof.* The formula for $t_{aver}(n, r)$ for any Revocation Scheme is:

$$t_{aver}(n, r) = \sum_{\substack{\mathcal{R}\subseteq\mathcal{N} \\ |\mathcal{R}|=r}} \frac{t(\mathcal{N}, \mathcal{R})}{\binom{n}{r}},$$

where $\mathcal{N}$ is the set of all users.

For any subset $\mathcal{R}$, $t(\mathcal{N}, \mathcal{R})$ can be expressed in terms of the nodes one level up from the leaves. Let $\mathcal{N}'$ be the set of nodes directly above the leaves. Let $S_a$, $S_s$ and $S_n$ be those nodes in $\mathcal{N}'$ with all, some and none of their children revoked respectively. If $\mathcal{R} = \emptyset$, then the root will have no children in $ST(\mathcal{R})$, and a corresponding index in the cover. Otherwise, the only nodes in $\mathcal{N}'$ corresponding to indices in the cover will be those nodes with some, but not all children revoked. Since $r \geq 0$, we have $\mathcal{R} \neq \emptyset$, and the only nodes in $\mathcal{N}'$ that have a corresponding index in the cover are those in $S_s$. The rest of the

cover will comprise of nodes that are ancestors of nodes in $\mathcal{N}'$ that have no child leaves revoked, i.e. $S_n$. Consequently, the size of the cover is equal to:

$$t(\mathcal{N}, \mathcal{R}) = |S_s| + t(\mathcal{N}', S_s \cup S_a) = |S_s| + t(\mathcal{N}', \mathcal{N}' \setminus S_n).$$

This, combined with Lemmas 35 and 36, gives:

$$\sum_{\substack{\mathcal{R} \subseteq \mathcal{N} \\ |\mathcal{R}| = r}} t(\mathcal{N}, \mathcal{R}) = \sum_{\substack{S_n \subseteq \mathcal{N}' \\ |S_n| \in r_1}} \sum_{\substack{S_s \subseteq \mathcal{N}' \setminus S_n \\ |S_s| \in r_2}} ns(r - a.Ar, Sr)(Sr + t(\mathcal{N}', \mathcal{N}' \setminus S_n)),$$

where $Ar = |S_a|$, $Sr = |S_s|$ and $Nr = |S_n|$. Each of these summations on the right hand side of the equation can be written as two separate summations. For the first one, we first sum $Nr$ over $r_1$, and then sum over all subsets $S_n \subseteq \mathcal{N}'$ of size $Nr$. Similarly for the second, we sum $Sr$ over $r_2$, and then sum over all subsets $S_s \subseteq \mathcal{N}' \setminus S_n$ of size $Sr$. However, nothing inside the summation depends on the actual subset $S_s$, just the size $|S_s| = Sr$. So we can replace this summation over $S_s \subseteq \mathcal{N}'$ with $\binom{n/a - Nr}{Sr}$:

$$\begin{aligned}
&\sum_{\substack{S_n \subseteq \mathcal{N}' \\ |S_n| \in r_1}} \sum_{\substack{S_s \subseteq \mathcal{N}' \setminus S_n \\ |S_s| \in r_2}} ns(r - a.Ar, Sr)(Sr + t(\mathcal{N}', \mathcal{N}' \setminus S_n)) \\
&= \sum_{Nr \in r_1} \sum_{\substack{S_n \subseteq \mathcal{N}' \\ |S_n| = Nr}} \sum_{Sr \in r_2} \sum_{\substack{S_s \subseteq \mathcal{N}' \setminus S_n \\ |S_s| = Sr}} ns(r - a.Ar, Sr)(Sr + t(\mathcal{N}', \mathcal{N}' \setminus S_n)) \\
&= \sum_{Nr \in r_1} \sum_{\substack{S_n \subseteq \mathcal{N}' \\ |S_n| = Nr}} \sum_{Sr \in r_2} \binom{n/a - Nr}{Sr} ns(r - a.Ar, Sr)(Sr + t(\mathcal{N}', \mathcal{N}' \setminus S_n)) \\
&= \sum_{\substack{Nr \in r_1 \\ Sr \in r_2}} \sum_{\substack{S_n \subseteq \mathcal{N}' \\ |S_n| = Nr}} \binom{n/a - Nr}{Sr} ns(r - a.Ar, Sr)(Sr + t(\mathcal{N}', \mathcal{N}' \setminus S_n)).
\end{aligned}$$

Only the $t(\mathcal{N}', \mathcal{N}' \setminus S_n))$ term depends on $S_n$, so we can take the rest out of the inner most summation. And using the earlier observation on the formula

for $t_{aver}(n, n-r)$ we get:

$$\begin{aligned}
&\sum_{\substack{Nr\in r_1\\ Sr\in r_2}} \sum_{\substack{S_n\subseteq\mathcal{N}'\\ |S_n|=Nr}} \binom{n/a-Nr}{Sr} ns(r-a.Ar, Sr)(Sr + t(\mathcal{N}', \mathcal{N}'\setminus S_n)) \\
&= \sum_{\substack{Nr\in r_1\\ Sr\in r_2}} \binom{n/a}{Nr}\binom{n/a-Nr}{Sr} ns(r-a.Ar, Sr)Sr \\
&\qquad\qquad + \binom{n/a-Nr}{Sr} ns(r-a.Ar, Sr) \sum_{\substack{S_n\subseteq\mathcal{N}'\\ |S_n|=Nr}} t(\mathcal{N}', \mathcal{N}'\setminus S_n) \\
&= \sum_{\substack{Nr\in r_1\\ Sr\in r_2}} \binom{n/a}{Nr}\binom{n/a-Nr}{Sr} ns(r-a.Ar, Sr)Sr \\
&\qquad\qquad + \binom{n/a-Nr}{Sr} ns(r-a.Ar, Sr)\binom{n/a}{Nr} t_{aver}\left(\frac{n}{a}, \frac{n}{a}-Nr\right) \\
&= \sum_{\substack{Nr\in r_1\\ Sr\in r_2}} \binom{n/a}{Nr\ Sr\ Ar} ns(r-a.Ar, Sr)\left(Sr + t_{aver}\left(\frac{n}{a}, \frac{n}{a}-Nr\right)\right).
\end{aligned}$$

As this is $\sum t(\mathcal{N}, \mathcal{R})$, in order to get $t_{aver}(n, r)$ we just have to divide by $\binom{n}{r}$. Therefore:

$$t_{aver}(n, r) = \frac{\sum_{\substack{Nr\in r_1\\ Sr\in r_2}} \binom{n/a}{Nr\ Sr\ Ar} ns(r-a.Ar, Sr)\left(Sr + t_{aver}\left(\frac{n}{a}, \frac{n}{a}-Nr\right)\right)}{\binom{n}{r}}.$$

□

Formula (4.15) gives us another way to compare the bandwidth for the Complete Subtree Revocation Scheme on different trees. In Figure 4.8 we see three different schemes that have roughly the same size user set: $243 \sim 256$. As expected the binary tree has the highest average bandwidth for the whole range of $r$. But the quaternary tree gives a lower average bandwidth than the ternary tree for most values of $r$, even though the quaternary tree scheme has a larger user set. As $a$ gets larger and larger, the average bandwidth gets smaller and smaller. How far this could be done in a practical setting would be limited by the $2^a$ factor in storage. The problem of finding a satisfactory trade-off point will be dealt with in Chapter 7.

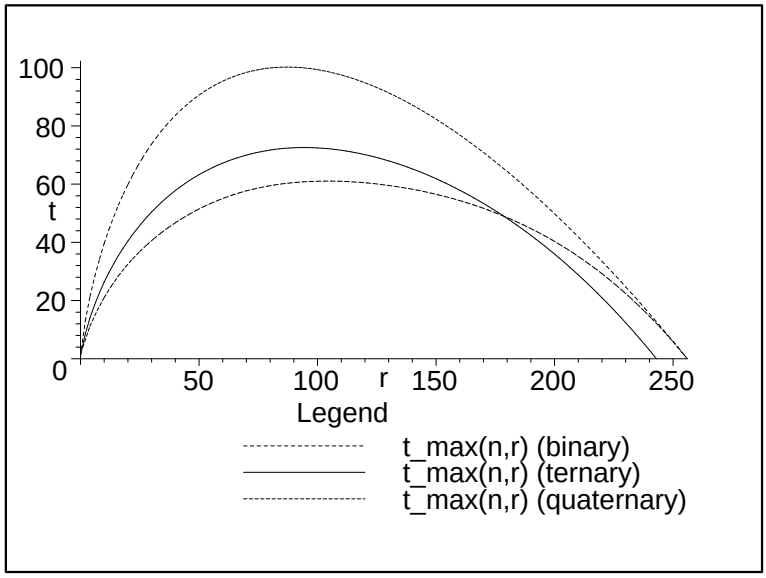


Figure 4.8: Top to bottom: $t_{aver}(n, r)$ for Complete Subtree Revocation Scheme with binary tree ($n = 2^8$), ternary tree ($n = 3^5$) and quaternary tree ($n = 4^4$).

### 4.3.3 Compression Method

Both methods of Asano go a long way to reducing the storage requirement in the $a$-ary tree based Complete Subtree Revocation Scheme, but place a heavy burden on the user. In Chapter 3 we saw how the user needs access to a large list of primes. One could argue that as the list of primes is public and would not require secure storage, the cost is insignificant. On the other hand, it may be necessary to limit all storage space at the receiver, for example if the receiver is a mobile device. The option of using a representation for the primes adds to the computational cost for key derivation by the users. Both Methods 1 and 2 require several multiplications and 1 modular exponentiation. It is worth noting that these modular exponentiations do not have the same cost in both Methods.

The computational complexity of raising $b^x \mod M$ is $\mathcal{O}(\log(x)\log^2(M))$. In Method 1 the exponent is the product of $(2^{a-1}-1)\log_a(n)$ primes of the order of $\mathcal{O}(2^a n \log(2^a n))$, giving an exponent with $\mathcal{O}(2^a \log(n)(\log(n)+a))$ bits. So the exponentiation will require $\mathcal{O}(2^a \log(n)(\log(n)+a)(\log^2(M)))$ operations. Method 2 uses a Master Key derived from fewer primes that are

smaller. The exponent is the product of just $2^{a-1}-1$ primes of size $\mathcal{O}(a)$, giving an exponent with $\mathcal{O}(2^a a)$ bits. The number of operations is still exponential in $a$: $\mathcal{O}(2^a a(\log^2(M)))$. The third method we propose has a much lower computational expense as well as other advantages.

**Method 3**

In Complete Subtree Revocation Scheme on an $a$-ary tree, each user has to store $(2^{a-1}-1)(\log_a(n))+1$ keys. This comes from $2^{a-1}-1$ keys for each node on the $a$-ary tree on the path from their leaf, $v_l$, to the root ($2^{a-1}-1$ being the number of proper subsets of $a$ children of a node, where each subset contains the ancestor of $v_l$). The Master Keys must be generated in such a way as to allow a user to calculate a key for a particular node $v_i$ and bit string $B$, if and only if the child of $v_i$ that is an ancestor of $v_l$ corresponds to a 1 in $B$ (naturally $v_i$ must be an ancestor of $v_l$ also). Method 1 uses a single Master Key for each user to generate their keys. Method 2 uses $\log_a(n)$ Master Keys, one for each node on the path from the leaf to the root, to generate the keys. Our Method 3 will generate the keys in the third logical manner: there are individual Master Keys for each bit string $B$, and these generate all $\log_a(n)$ keys on the path from the leaf to the root.

The calculations involved are not the same as the method of Asano, although they do use $RSA$ type calculations. The two methods of Asano allow establishment keys to be generated directly from the Master Key(s). With the third method, the Master Keys are assigned to the leaves of the user they belong to. The centre publishes a function (which we will describe shortly) that generates a key for a particular node from one of its child's keys. A user can then generate the key for any node on the path from his leaf to the root by generating all intermediate keys. In this sense, this third compression method has more in common with the use of the Pseudo-Random Number Generator to create labels for the Subset Difference Revocation Scheme (Section 3.3.2). An important difference is that with the latter one of the nodes moves further from the root with each use of the function. Before describing the method, we will describe the requirements of this function.

The first requirement is that it be invertible. This is an important difference to the method of $SDRS$, and it stems from how the Master Keys will be used. In $SDRS$, Master Keys (or labels) are associated with internal nodes. Users apply a function ($PRNG$) to get labels further down the tree. With our method, users will be given Master Keys associated with their leaves, and apply a function to get keys associated with nodes higher up the tree. These keys will have to be the same for different users, by nature of the fact that they are shared keys. In order to create keys with this property, the centre must start with the shared key and apply the inverse of the function the users apply. We will use the following notation: the Master Key for the bit string $B$ assigned to user $u$ is $MK_{l,B}$ (where $f(u) = v_l$ or $v_l$ is the leaf $u$ is assigned). Establishment keys are indexed by both node and bit string: $L_{i,B}$ is the establishment key for node $v_i$ and bit string $B$. The intermediate keys that users generate by applying the function to their Master Keys we will label $IK(u, v_i, B)$ (this is generated by user $u$ and corresponds to the node $v_i$ in the tree). The centre will start at the root and work down, while the users will have Master Keys corresponding to leaves and work up. We will call the function used by the centre in generating the Master Key:

$$MOVE\_DOWN(IK(u, v_i, B)) = IK(u, child(v_i), B),$$

and the function used by the users to generate establishment keys:

$$MOVE\_UP(IK(u, v_i, B)) = IK(u, parent(v_i), B).$$

Since (some of) the intermediate keys will be available to the users, we need to ensure $MOVE\_DOWN()$ remains secret. As $MOVE\_DOWN()$ is the inverse of $MOVE\_UP()$ (which the users have), what we need is a one-way trapdoor function. Hence the use of $RSA$ calculations. However, the multiple paths requiring different keys make things more complicated.

Consider the intermediate key of two siblings $v_{i_1}$ and $v_{i_2}$. Establishment keys for the Complete Subtree on an $a$-ary tree are only known to a subset of users of the nodes they are defined on (the subset defined by the bit string $B$). If $v_{i_1}$ and $v_{i_2}$ are distinct nodes, they will have distinct descendants. In order for the establishment keys for $v_{i_1}$ and $v_{i_2}$ to be known

only by the subsets of descendants of the respective nodes, $IK(u_1, v_{i_1}, B)$ must not be equal to $IK(u_2, v_{i_2}, B)$, nor can one be calculated from the other using $MOVE\_DOWN()$. So $MOVE\_DOWN()$ will generate different values for different children of the same node. As a consequence of this, $MOVE\_DOWN()$ for different paths cannot commute. For example, if we start with the intermediate key for $v_i$ and then use $MOVE\_DOWN()$ to work out the intermediate key of the first child of the second child of $v_i$, it cannot give the same value as the intermediate key of the second child of the first child of $v_i$. You do not get to the same node in a tree by taking the same steps in a different order. The first child of the second child of $v_i$ is not the same node as the second child of the first child of $v_i$. These nodes have different descendants and so must have different intermediate keys.

Consider also the intermediate key of a node $v_i$ that is the parent of two nodes $v_{i_1}$ and $v_{i_2}$, where $v_{i_1}$ corresponds to a 1 in $B$ and $v_{i_2}$ corresponds to a 0. Any user $u_{j_1}$ whose leaf is a descendant of $v_{i_1}$, can use one of their Master Keys to generate an intermediate key for $v_i$, as can any user $u_{j_2}$ with leaf descended from $v_{i_2}$. However, by the definition of the two nodes $v_{i_1}$ and $v_{i_2}$, $u_{j_1} \in \gamma(i, B)$ but $u_{j_2} \notin \gamma(i, B)$. Therefore, $MOVE\_DOWN()$ must generate two different values for the intermediate key of a given node, one for users in $\gamma(i, B)$ and one for users not contained in $\gamma(i, B)$. $L_{i,B}$ would only be generated from $IK(u, v_i, B)$ where $u \in \gamma(i, B)$. The actual value of $L_{i,B}$ is found by taking an appropriate hash of $IK(u, v_i, B)$ for an added level of protection. We will now define these two functions for our compression method.

As the scheme is defined, the centre will have an $a$-ary tree, labelled with breadth first order. The users are assigned to the leaves of the tree. For reasons that will be apparent later, the users' leaves must all be descended from a single child of the root. This essentially means adding an extra edge to the root of a complete tree (see Figure 4.9). The centre must choose the primes $q_1$ and $q_2$ for the public modulus, i.e. $M = q_1 q_2$. For generating the Master Keys, it only needs $a$ primes $p_1, p_2, \ldots, p_a$, but they each must be co-prime to $\varphi(M)$. The centre needs to be able to calculate the list $d_i = p_i^{-1} \mod \varphi(M), i = 1, \ldots, a$. The primes $p_i$ and the modulus $M$ are made public. We define the two functions as follows: Let $v_i$ be the $j^{th}$ child of $parent(v_i)$

in the $a$-ary tree. Let $v_{i'}$ be a child of $v_i$ such that user $u$ is assigned to a leaf that is a descendant of $v_{i'}$. Then:

$$IK(u, v_i, B) = MOVE\_DOWN(IK(u, parent(v_i), B))$$
$$= \begin{cases} (IK(u, parent(v_i), B) + 1)^{d_j} \mod M \\ \quad \text{if } u \in \gamma(v_{i'}, B) \\ (IK(u, parent(v_i), B) + 2)^{d_j} \mod M \\ \quad \text{otherwise} \end{cases}$$

The definition of $MOVE\_UP()$ follows directly from the above:

$$IK(u, parent(v_i), B) = MOVE\_UP(IK(u, v_i, B))$$
$$= \begin{cases} (IK(u, v_i, B)^{p_j} \mod M) - 1 \\ \quad \text{if } u \in \gamma(v_{i'}, B) \\ (IK(u, v_i, B)^{p_j} \mod M) - 2 \\ \quad \text{otherwise} \end{cases}$$

Note that to generate the intermediate key for a node $v_i$ we use the relation ship between $v_i$ and its child, $v_{i'}$, (to decide if we add 1 or 2) and the relationship between $v_i$ and its parent (to pick which exponent we use). The intermediate keys for the root will be chosen randomly, and all other keys will be generated from these seeds by applying $MOVE\_DOWN()$.

Although $u$ is an argument of $MOVE\_DOWN()$, the centre does not need to perform all the calculations to generate a Master Key separately for each user. There are only two different values for an intermediate key for any node, one for users contained in $\gamma(i, B)$ and one for those who are not. That means that the centre need only generate $2(n-1)$ intermediate keys (twice the number of internal nodes), as compared to $n \log_a(n)$ if working out each Master Key separately. As well as giving two different values for each node as required, this definition of $MOVE\_DOWN()$ ensures that the intermediate keys of siblings will also be different. The value of $IK(u, v_i, B)$ depends on the relationship between $v_i$ and its parent. Since this will be different for any sibling of $v_i$, $IK(u, sibling(v_i), B)$ will be different as well (a different power is used in the exponentiation).

| | **Algorithm for the Generation of Master Key $MK_{l,B}$** |
|---|---|
| | Gen_MK($u$,$B$,$M$,$d$,$K_B$):= |
| 0: | $f(v_l) := u$ |
| 1: | $v_i :=$ first (and only) child of the root |
| 2: | $IK(u, v_i, B) = \begin{cases} (K_B + 1)^{d_1} \mod M & \text{if } u \in \gamma(i, B) \\ (K_B + 2)^{d_1} \mod M & \text{otherwise} \end{cases}$ |
| 3: | **while** $v_i \neq parent(v_l)$ **do** |
| 4: | $v_i :=$ child of $v_i$ that is ancestor of $v_l$ |
| 5: | $IK(u, v_i, B) := MOVE\_DOWN(IK(u, parent(v_i), B))$ |
| 6: | **end do** |
| 7: | $MK_{l,B} := IK(u, parent(v_l), B)$ |
| 8: | return($MK_{l,B}$) |

Table 4.3: Master Key Generation in Method 3.

Normally $RSA$ encryption does commute, it does not matter which order you use the exponents you get the same result. However, the addition of +1 (or +2) before exponentiation means this no longer holds. For example, if $M = 17 \times 23$ and $p_1 = 3$, $p_2 = 5$:

$$\begin{aligned} ((20+1)^3+1)^5 &\equiv 29 \mod M, \\ ((20+1)^5+1)^3 &\equiv 40 \mod M, \\ \text{whereas} \quad (20^3)^5 &\equiv (20^5)^3 \equiv 57 \mod M. \end{aligned}$$

Now we can describe how the Master Keys are generated. The centre randomly chooses $2^a - 2$ elements $K_B \mod M$, where $B = b_1 b_2 \ldots b_a$, $b_i \in \{0, 1\}$ and $\sum_{i=1}^{a} b_i \neq 0$ or $a$. These are the intermediate keys corresponding to the root for each bit string $B$. The Master Keys are generated by repeatedly applying $MOVE\_DOWN()$. The algorithm to generate the Master Key for subset $B = b_1 b_2 \ldots b_a$ and user $u$ ($f(v_l) = u$) is given in Table 4.3.

The important components of $MOVE\_DOWN()$, the exponents $d_i$, are known only to the centre. But the exponents of $MOVE\_UP()$ are the public primes. Along with their Master Key, any user can use the algorithm in Table 4.4 to calculate any establishment key $L_{i_1,B}$, where $v_{i_1}$ is on the path from their leaf to the root.

| | **Algorithm for Generating Establishment Keys** |
|---|---|
| 0: | Gen_SK($v_l$,$B$,$MK_{l,B}$,$M$,$p_1,\ldots,p_a$) |
| 1: | $v_i := par(v_l)$ |
| 2: | $IK(u,i,B) := MK_{l,B}$ |
| 3: | **while** $v_i \neq v_{i_1}$ **do** |
| 4: | $IK(u,parent(v_i),B) := MOVE\_UP(IK(u,v_i,B))$ |
| 5: | $v_i := parent(v_i)$ |
| 6: | **end do** |
| 7: | $L_{i,B} = Hash(IK(u,v_{i_1},B))$ |
| 7: | return($L_{i,B}$) |

Table 4.4: Secret Key Generation in Method 3

The algorithm is simply reversing the operations of the Master Key generating algorithm. Raising a number to the power of any of the numbers in the list $d_i$ is undone by raising to the power of the equivalent prime $p_i$. The extra term $\{1,2\}$ that was added in the generation phase is simply subtracted.

Suppose a user wishes to generate an establishment key for an set $\gamma(i,B)$ to which they do not belong. If they have an intermediate key for $v_i$, then the value they have is in the form $(x+2)^{d_j} \mod M$, while the intermediate key that generates $L_{i,B}$ is $(x+1)^{d_j} \mod M$. All they can do with the public information is work out $x$, but working out $(x+1)^{d_j} \mod M$ from $x$ requires either the secret exponent $d_j$ or factoring $M$. If they do not have an intermediate key for the node $v_i$, then they will have the key of an ancestor of $v_i$ which was used to generate $IK(u,i,B)$. But generating $IK(u,i,B)$ from this ancestor key also requires knowledge of the secret exponents or factors of $M$.

Figure 4.9 illustrates the Master Key generation process for a small user set ($a = 3$, $n = a^2 = 9$) and bit string $B = 110$. Both intermediate keys are shown for each internal node. The Master Key for any particular user will be one of the intermediate keys of its parent, for this value of $B$ it will be the upper key (as they appear in Figure 4.9) for the first two children of the parent and the lower key for the third. This means that some users will share Master Keys. This is something that will happen for particular bit strings, e.g. $u_1$ and $u_2$ belong to $\gamma(2,B) = \{u_1\}$ ($v_2$ is their parent) and $\gamma(1,B) = \{u_1,u_2,u_3,u_4,u_5,u_6\}$. However, the set of Master Keys for a given

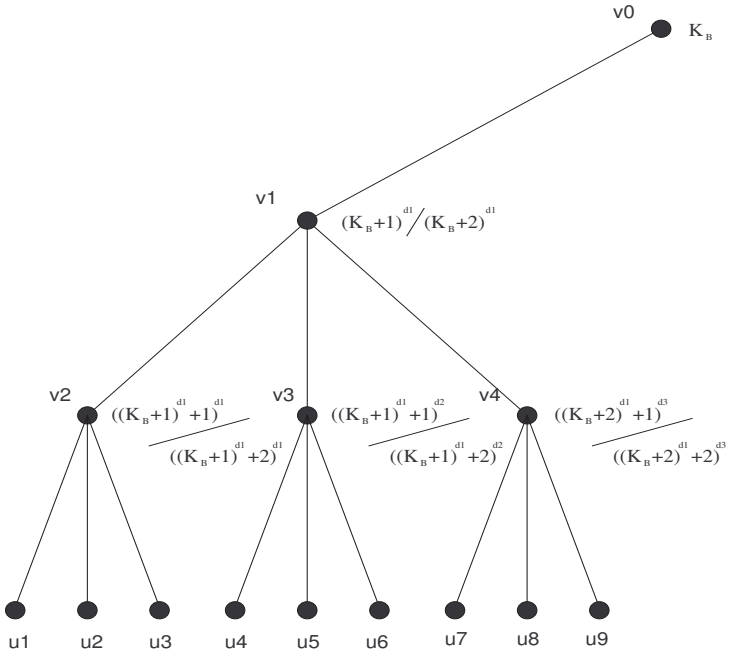


Figure 4.9: Example of Compression Method 3.

user (there will be one for each $2^a - 2$ bit strings) will be unique.

As we have said, $u_1$ and $u_2$ receive $((K_B + 1)^{d_1} + 1)^{d_1} \mod M$ as Master Keys. The establishment key $L_{2,B}$ is just the hash of this value. User $u_3$ cannot calculate this as his Master Key is $((K_B + 1)^{d_1} + 2)^{d_1} \mod M$. He can calculate $(K_B+1)^{d_1} \mod M$ using the public prime $p_1$. By checking the figure we can see that all of $u_1, \ldots, u_6$ can calculate this value, which is exactly the set $\gamma(1, B)$. The last three users cannot calculate this value as raising to the power of $p_3$ will give $(K'_B + 1)^{d_1} \mod M$. The only establishment key any of these three users can calculate is $L_{4,B}$ which is the hash of $((K_B + 2)^{d_1} + 1)^{d_3}$, and just like before, $u_9$ cannot calculate this from his Master Key.

If we were to extend this to a tree with 27 users, we would see why $MOVE\_DOWN()$ must not commute. One of the keys for the node currently the leaf for $u_2$ would be $(((K_B + 1)^{d_1} + 1)^{d_1} + 1)^{d_2} \mod M$. The node that is currently the leaf for $u_4$ would have $(((K_B+1)^{d_1}+1)^{d_2}+1)^{d_1} \mod M$. If $MOVE\_DOWN()$ commuted, then these two intermediate keys would be the same, even thought they are for completely different nodes.

The above process of generating Master Keys must be repeated for all

$2^a - 2$ bit strings $B$ (where $B$ is neither all 0's nor all 1's). However, it can happen that several users are given Master Keys that do not generate any establishment keys. Consider the Master Key for $u_9$ in the example in Figure 4.9. Since $B = 110$, $u_9$ belongs to neither $\gamma(4, B)$ nor $\gamma(1, B)$. We have designed the method so $u_9$ cannot generate either establishment keys from their Master Key (nor any other establishment key). As this results in a Master Key that serves no purpose for the user, it need not be given to him, and so gives a slight reduction in storage. This would only happen for very specific paths from the root. If all ancestors of a given leaf were children corresponding to the zero bits of $B$, then the Master Key for that leaf would be redundant. If the depth of the tree ($\log_a(n)$) was greater than $a$, then there would always be at least one user with no redundant Master Keys . Since each bit string must have at least one non-zero bit the $1^{st}$ child of the $2^{nd}$ child of the ... of the $a^{th}$ child of any node in the tree will generate at least one secret key from any of its Master Keys. In this case at least one user stores $2^a - 2$ Master Keys, i.e. no Master Keys are redundant. We do not use the $|U|_{\max}$ notation to represent this as we are not counting establishment keys, but instead we say the maximum storage for any user is $2a - 2$ Master Keys. If $\log_a(n) < a$, each user will have at least $2^{a-\log_a(n)} - 1$ redundant Master Keys. The path from any leaf will have at most $\log_a(n)$ different relations between a node and it's parent. If all these relations correspond to zero in $B$ the key is redundant. There are $2^{a-\log_a(n)} - 1$ such bit strings $B$ (have to exclude the all zero string). This gives a slight reduction to $2^a - 2 - (2^{a-\log_a(n)} - 1) = 2^a - 2^{a-\log_a(n)} - 1$ Master Keys.

There is still the matter of the extra key for the set of all users. There is already a key that can be used for this set. Any user can use $MOVE\_UP()$ to work back all the way up to $K_B$. In particular $K_B$, where $B = 10 \ldots 0$, would be interpreted as the intermediate key for the users whose leaves are descended from the first child of the root. But as we stipulated that all leaves be descended from this node, this is all the users. So defining the intermediate key $IK(root, 10 \ldots 0)$ to be $K_{10\ldots0}$ means all users can generate this key. There is no extra storage cost associated with this key as users are just using one of the existing keys. In fact, besides the extra computation, there is no cost

for users generating keys several levels above the ancestor of all leaves. This means the centre can use a tree several times bigger than necessary without adding an extra storage burden to the users. It just means adding a path of several edges to the root of a complete $a$-ary tree instead of just one as in Figure 4.9. This allows for adding more users to the scheme later without needing to update existing users.

The decision of which of the three methods to use depends largely on the capabilities of the receivers. We have already discussed the large list of primes needed for Methods 1 and 2, and the computation needed if a representation is used instead. We will further analyse the computational complexity of the modular exponentiation in all three methods. If it turns out that the requirements of Method 1 are not prohibitive, then it clearly is the most advantageous method as each user only has 1 Master Key. We know that $n$ is fixed by the population of users in the scheme. Suppose that $a$ is also fixed in order to limit the bandwidth. With Method 1, each user must be capable of the following:

- Storage of $(2^{a-1}-1)\log_a(n)+1$ primes of size $\mathcal{O}(2^a n \log(2^a n))$ (roughly $\mathcal{O}(2^a \log(n)(\log(n)+a))$ bits)
- Storage of one Master Key
- Multiplication of $(2^{a-1}-1)\log_a(n)$ primes
- Exponentiation $\mod M$ with exponent of $\mathcal{O}(2^a \log(n)(\log(n)+a))$ bits

By using a representation of the primes, the user need only store one integer $X$, which is $\mathcal{O}(a+\log(a+\log(a\log(n))))$ bits long, but must also do $\mathcal{O}(\frac{2^a \log^5(n)}{\log(a)})$ extra operations in generating primes.

If these criteria cannot be met, the more appropriate of the other two methods will vary depending on the values of $n$ and $a$. For Method 2, each user must be capable of the following:

- Storage of $2^{a-1}$ primes of size $\mathcal{O}((2^a-1)\log(2^a-1))$ (roughly $\mathcal{O}(2^a(a+\log(a)))$ bits)

- Storage of $\log_a(n)$ Master Keys
- Multiplication of $2^{a-1} - 1$ primes
- Exponentiation $\mod M$ with exponent of $\mathcal{O}(2^a(a + \log(n)))$ bits

Finally, Method 3 has the following requirements:

- Storage of $a$ primes of size $\mathcal{O}(a \log(a))$ (roughly $\mathcal{O}(a \log(a))$ bits)
- Storage of $2^a - 2$ Master Keys
- No multiplications
- $\log_a(n)$ exponentiations $\mod M$ with exponent of $\mathcal{O}(\log(a))$ bits, which is equivalent to one exponentiation with exponent of $\mathcal{O}(\log(n) \log(a))$ bits

In comparing Methods 2 and 3, we see that in terms of storing primes, multiplications and exponentiations, Method 3 has lesser requirements. The only area where Method 2 could, and for the most part does, perform better than Method 3 is in the number of Master Keys to be stored. While it is true that $\log_a(n)$ will be less than $2^a - 2$ for most values of $a$ and $n$, there are some non-trivial parameter values with $2^a - 2 < \log_a(n)$. If $a = 2$ (there is no reason why any of the compression methods cannot be applied to the Complete Subtree Revocation Scheme on a binary tree), then Method 3 has the lesser storage for all $n \geq 8$. If $n \geq 8$, then $\log_2(n) \geq 2 = 2^2 - 2$. For the ternary case, Method 3 requires fewer Master Keys when $n \geq 2187$. This is a reasonably small value for $n$, it is conceivable that there could be uses for a scheme with this many users. Once we get to $a = 4$ and higher, Method 2 will require fewer Master Keys for all values of $n$ that matter. We would need $n$ to be greater than $1.1 \times 10^9$ for Method 3 to perform better (in the $a = 4$ case). Seeing as this is a sixth of the world population, it is unlikely that a scheme this large would be needed. Even if the parameters are such that Method 3 requires more Master Keys, the lesser burden on the user with regard the storage of primes and various calculations could make Method 3 the more efficient method.

In this chapter we have given a formula for $t_{\max}(n, r)$ for the Complete Subtree, an improvement on the existing bound. We have also used a recursive formula to work out $t_{aver}(n, r)$ for large values of $n$. We generalised these formula to the Complete Subtree Revocation Scheme on an $a$-ary tree, which results in schemes with significantly lower bandwidth costs. Finally, we defined a third compression method to reduce the large storage costs of these schemes, and showed how this method compliments the existing two schemes. In Chapter 5 we will look at a different variation on the Complete Subtree, and perform a similar analysis of its performance.

# Chapter 5

# Forest of Trees Revocation Scheme

In this chapter we will discuss some possible improvements on the Complete Subtree Revocation Scheme. We describe some general methods for combining different schemes and discuss some of the properties of the resulting schemes. We give examples of constructions of schemes using these methods that have some desirable qualities: in this case, a measurably lower $t_{\max}(n, r)$ than the Complete Subtree Revocation Scheme, but with slightly higher $|U|_{\max}$. First, we will describe the different schemes that we will use.

When we defined the Complete Subtree Revocation Scheme, we used a function $f$ that formed a one-to-one mapping from the set of leaves of the binary tree $T$ to the set of users $\mathcal{N}$. The only $f$ we used was $f(v_i) = u_{i-(n-1)}$, as it was the simplest. We could have instead used $f'(i) = \pi(f(v_i))$, where $\pi$ is a permutation on $\mathcal{N}$. This would have given rise to a scheme with the same general properties, i.e. $t_{\max}(n, r)$, $t_{aver}(n, r)$ and $|U|_{\max}$ would all be the same. But the sets of users who shared the same key would be different (the same index $i$ would give two different sets $\gamma_f(i)$ and $\gamma_{f'}(i)$). On their own both schemes are the same, but we hope to find a way to combine such schemes to obtain some improvement over the Complete Subtree Revocation Scheme.

Obviously, we will need a way to represent the various schemes, ideally one that is more compact than listing all the inputs/outputs to $\gamma_{f'}$. We will keep the same complete binary tree $T$ (and the same breadth first labelling), as well

as the function $f$. This means that the leaves of $T$ (the inputs to $f$) will always be $v_n, \ldots, v_{2n-1}$. We will use the permutation $\pi$ on $\mathcal{N}$ to differentiate between the functions $\gamma_{f'}$. In order to represent the scheme clearly and succinctly, we will use the following representation:

*Definition* 40. Let $\mathcal{N}$ be a set of $n$ users. Let $CSRS = (\mathcal{N}, \Omega, \gamma_{f'})$ be a Complete Subtree Revocation Scheme on the complete binary tree $T$ with breadth first labelling $\{v_1, \ldots, v_{2n-1}\}$. The *leaf list* of $CSRS$ is:

$$[f'(v_i) : i = n, \ldots, 2n-1].$$

Since the tree uses breadth first labelling, $[v_n, \ldots, v_{2n-1}]$ is a list of the leaves of the tree $T$. The purpose of $f'$ is to map leaves to users, so the leaf list is all the users listed in the order that they are assigned to leaves on the tree (reading from left to right). The advantage of this representation is that from just looking at the list we can see what sets of users share keys. For example, the Complete Subtree Revocation Scheme on $n = 2^3$ users, with $\gamma_f$ (where $f(v_i) = u_{i-(n-1)}$) would have the leaf list $[u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8]$. The singletons are always the same ($\{u_i\}, \forall u_i \in \mathcal{N}$). The sets of pairs of users who share a key, $\{u_1, u_2\}, \{u_3, u_4\}, \{u_5, u_6\}, \{u_7, u_8\}$, is found by splitting the leaf list evenly into 4. The sets of 4 users who share a key are $\{u_1, u_2, u_3, u_4\}, \{u_5, u_6, u_7, u_8\}$, and all users share one key assigned to the root. From this list alone, it is possible to generate all subsets $\gamma_f(i)$ in a Complete Subtree Revocation Scheme (the leaf list is only defined for this type of scheme). We just divide this list evenly into sets of size a power of 2, for all powers of 2 less than or equal to $n$.

We will define any other instance of the Complete Subtree Revocation Scheme to have the function $\gamma_{f^*}$, with

$$f^*(v_i) = \pi^*(f(v_i)),$$

where $\pi^*$ is a permutation on $\mathcal{N}$ and $f(v_i) = u_{i-(n-1)}$. Figure 5.1 shows two leaf lists, and the binary tree and subsets of users corresponding to both. The leaf lists correspond to the functions $f$ and $f^*(v_i) = \pi^*(f(v_i))$, where $\pi^* = (1)(4635872)$. Note that the trivial subsets (which equate to keys held by only one user) and the complete set (key held by all users) are always

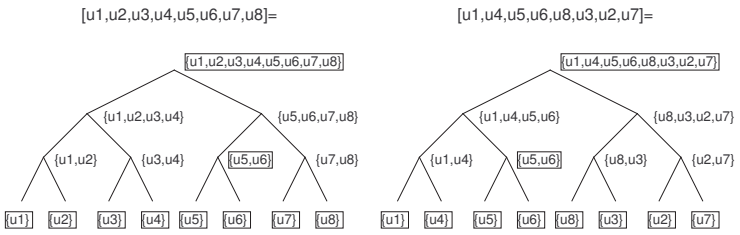


Figure 5.1: A Forest of two trees. Common subsets are contained in squares.

common to both trees. Any other subsets (non-trivial, proper) may or may not be common depending on the choice of permutation. In this example $\{u_5, u_6\}$ is common to both trees, but it is the only subset (besides those mentioned above) that is. None of $\{u_1, u_4\}, \{u_8, u_3\}, \{u_2, u_7\}$ occur in the first tree and all sets of size 4 differ.

## 5.1 Combining Schemes

In this section, we will describe two methods of combining Revocation Schemes to form new schemes. The first method requires that the component schemes all be defined on the same user set, and forms a new scheme on the same user set. The second method requires two Revocation Scheme with disjoint user sets, and forms a union of the schemes defined on the union of the disjoint user sets (and is for that reason called a *disjoint union*).

### 5.1.1 Union of Schemes

If we have two (or more) schemes that have the same user set and the same index set, then there is a very natural way to combine them:

*Definition* 41. Let $RS_1 = (\mathcal{N}, \Omega, \gamma_1), \ldots, RS_X = (\mathcal{N}, \Omega, \gamma_X)$ be $X$ Revocation Schemes. We define the *union* of these schemes to be $RS = (\mathcal{N}, \Omega', \gamma')$ where:

$$\begin{aligned} \Omega' &= \Omega \times \{1, \ldots, X\} \\ \gamma'(i,j) &= \gamma_j(i) \quad \text{where } i \in \Omega, 1 \leq j \leq X \end{aligned}$$

It is simple to prove that the union of Revocation Schemes is itself a Revocation Scheme.

*Lemma* 42. Let $RS$ be the union of the Revocation Schemes $RS_1, RS_2, \ldots, RS_X$, each of which have the user set $\mathcal{N}$. Then $RS$ is also a Revocation Scheme.

*Proof.* Since each component of the union is a Revocation Scheme, for any user $u \in \mathcal{N}$ we can find an index $i$ in any of the $X$ schemes such that $\gamma_j(i) = \gamma'(i,j) = \{u\}$. By the definition of a union, all such sets also occur in $RS$. Since the user set for $RS$ is also $\mathcal{N}$, all singletons of $\mathcal{N}$ occur in $RS$. Therefore $RS$ is a Revocation Scheme. □

In the example shown in Figure 5.1, we can clearly see all 8 singletons appearing in either tree. All subsets shown would be in the union of the two schemes. We can make the following statement about $t_{\max}(n,r)$ of the resulting scheme.

*Lemma* 43. Let $RS = (\mathcal{N}, \Omega', \gamma')$ be the union of the Revocation Schemes $RS_1 = (\mathcal{N}, \Omega, \gamma_1), \ldots, RS_X = (\mathcal{N}, \Omega, \gamma_X)$. Then:

$$t^{RS}_{\max}(n,r) \leq \min(t^{RS_1}_{\max}(n,r), \ldots, t^{RS_X}_{\max}(n,r)). \tag{5.1}$$

*Proof.* Let $\mathcal{R}$ be any subset of $\mathcal{N}$ of size $r$. By the definition of $t_{\max}(n,r)$ we have that:

$$t^{RS_1}(\mathcal{N}, \mathcal{R}) \leq t^{RS_1}_{\max}(n,r).$$

Let $S$ be the minimal cover of $\mathcal{N} \setminus \mathcal{R}$ in $RS_1$ ($|S| = t^{RS_1}(\mathcal{N}, \mathcal{R})$). For every set $s \in S$ there is some index $j$ in $\Omega$ such that $\gamma_1(j) = s$. But we also have that $\gamma_1(j) = \gamma'(1,j)$ from the definition of the union. Therefore $S$ is also a cover of $\mathcal{N} \setminus \mathcal{R}$ in $RS$. It may not be the minimal cover in $RS$, so we can only say that:

$$\begin{aligned} t^{RS}(\mathcal{N}, \mathcal{R}) &\leq |S| \\ &= t^{RS_1}(\mathcal{N}, \mathcal{R}) \\ &\leq t^{RS_1}_{\max}(n,r). \end{aligned}$$

Since this is true for all $\mathcal{R} \subseteq \mathcal{N}$, it holds for $\mathcal{R}$ such that $t^{RS}(\mathcal{N}, \mathcal{R}) = t^{RS}_{\max}(n,r)$. Therefore:

$$t^{RS}_{\max}(n,r) \leq t^{RS_1}_{\max}(n,r).$$

This holds for all the Revocations schemes in the union, therefore:

$$t^{RS}_{\max}(n,r) \leq \min(t^{RS_1}_{\max}(n,r), \ldots, t^{RS_X}_{\max}(n,r)).$$ □

We could have defined a union of schemes with different index sets, but for what we will be trying to achieve later, we will need the index sets to be the same. The definition is not limited to the Complete Subtree Revocation Scheme, any collection of Revocation Schemes can be used, provided they have the same user and index set. However, we will only be looking at the unions of Complete Subtree Revocation Schemes. We will call the union of several different Complete Subtree Revocation Schemes a *Forest of Trees* Revocation Scheme. Strictly speaking there is only one tree, and several permutations on the assignments of users to the leaves, but it sometimes helps to consider the different trees generated as in Figure 5.1. These are trees where each node $v_i$ is labelled with the set of users $\gamma(i)$. The following example shows one advantage of combining two schemes that are essentially the same.

*Example* 44. Suppose we wanted to revoke the set $\mathcal{R} = \{u_4, u_5, u_6, u_7\}$, using the schemes in Figure 5.1. We need to find a cover of $\mathcal{N} \setminus \mathcal{R} = \{u_1, u_2, u_3, u_8\}$ using only the subsets in the diagram. With just the first tree we get a cover of $\{\{u_1, u_2\}, \{u_3\}, \{u_8\}\}$. The second tree does not do any better own its own, $\{\{u_1\}, \{u_2\}, \{u_3, u_8\}\}$. The size of the minimum cover is 3 in both cases. However, in union of these two schemes, $t(\mathcal{N}, \mathcal{R})$ is only 2. The minimal cover is $\{\{u_1, u_2\}, \{u_3, u_8\}\}$.

The size of the minimal cover in the combined scheme is not the minimum of the covers in the two component schemes. Similarly, $t_{\max}(n, r)$ in a union of schemes is not always the minimum of $t_{\max}(n, r)$ in each individual scheme. We will see an example of this later.

Suppose we have a union of Revocation Schemes. Assuming that we can generate a cover of any subset of users in each individual scheme (which we can for the Complete Subtree Revocation Scheme), how can we find a cover for the same subset in the larger scheme?

### 5.1.2 Greedy Algorithm

Table 5.1 contains an explicit algorithm for finding a cover with a union of schemes (not limited to a Forest of Trees), $RS_1 = (\mathcal{N}, \Omega, \gamma_1)$, ..., $RS_X = (\mathcal{N}, \Omega, \gamma_X)$. This algorithm first pools together all subsets of privileged users

| | **Cover Algorithm** |
|---|---|
| 0: | **Initialise:** $S' = \{\}, S = \{\}, P = \mathcal{N} \setminus \mathcal{R}$ |
| 1: | **for** $i$ from 1 to $X$ **do** |
| 2: | Find all sets $\gamma_i(j)$ in scheme $RS_i$ that are contained in $P$ and add to set $S'$ ($S' = S' \cup \gamma_i(j)$) |
| 3: | **end do** |
| 4: | **while** $\bigcup_{s \in S} s$ does not cover $\mathcal{N} \setminus \mathcal{R}$ **do** |
| 5: | Find the set $s \in S'$ of largest cardinality such that $s \subseteq P$ and add it to $S$ ($S = S \cup s$) |
| 6: | Subtract $s$ from $P$ ($P = P \setminus s$) |
| 7: | Subtract $s$ from $S'$ ($S' = S' \setminus \{s\}$) |
| 8: | **end do** |

Table 5.1: Algorithm to find the cover in a union of schemes.

from the various schemes into the set $S'$. These are the only subsets that can be used in the cover as they are strictly contained in $\mathcal{N} \setminus \mathcal{R}$. The set $P$ is defined to be the set of users who have not yet been covered. The algorithm repeatedly adds the largest set from $S'$ that is contained in $P$ to $S$, until $S$ is a cover of $\mathcal{N} \setminus \mathcal{R}$.

The algorithm is only of use if it terminates for all input $\mathcal{R}$. For any of the $X$ Revocation Schemes (named $RS_i$), we know that there exists some $j$ such that $\gamma_i(j) = \{u\}$, for any $u$. This means that $\gamma_i(j) = \{u\} \in S'$ for all $u \in P$ at the end of the first **for** loop (i.e. every singleton with a privileged user will be in $S'$). So in the body of the **while** loop, at the very least we can add a singleton to $S$. Therefore the loop will take at most $|\mathcal{N} \setminus \mathcal{R}|$ steps before finishing. When it does finish, $S$ is a cover of $\mathcal{N} \setminus \mathcal{R}$, as this is the terminating condition. Additionally, the cover is also disjoint. Each set $s$ chosen to be in the cover must be strictly contained in the set of those privileged users who have not yet been placed in the cover. So the set $s$ cannot intersect with any set that has already been added.

One point of note is that step 2 finds all subsets of privileged users from the current scheme $RS_i$, and not a cover with $RS_i$, which is what you might expect. A cover of the privileged users will certainly be contained in the former, but we need the extra subsets in order to be able to form a disjoint

cover. The different covers from the different schemes may well be individually disjoint, but since they all cover the same set of users, they will overlap with each other. We can run into problems when trying to combine subsets that do overlap into a cover that does not. This is best illustrated with an example:

*Example* 45. Suppose we want to use the two trees in Figure 5.1 to revoke $\mathcal{R} = \{u_2, u_3\}$, but only using a disjoint cover. The first tree gives a cover of $\{\{u_1\}, \{u_4\}, \{u_5, u_6, u_7, u_8\}\}$, while the second tree gives us the cover $\{\{u_1, u_4, u_5, u_6\}, \{u_8\}, \{u_7\}\}$. Now both covers are of size 3, and given that we cannot uses overlapping sets in the cover, there is no way of generating a smaller cover with just these sets. But the greedy algorithm above will also generate the extra subsets $\{u_1, u_4\}, \{u_5, u_6\}$ and $\{u_5, u_6\}, \{u_7, u_8\}$ (as well as the singletons $\{u_5\}, \{u_6\}$). Using these we can form a cover of size 2: either $\{\{u_1, u_4\}, \{u_5, u_6, u_7, u_8\}\}$ or $\{\{u_1, u_4, u_5, u_6\}, \{u_8, u_7\}\}$

This example shows that the greedy algorithm can produce a smaller cover than just using the minimum cover of the different schemes. Unfortunately, this does not always happen:

*Example* 46. Suppose $\mathcal{N} \setminus \mathcal{R} = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$, and we have two different coverings from two different schemes:

$$\begin{aligned} c_1 &= \{\{u_1, u_2, u_3, u_4\}, \{u_5\}, \{u_6\}, \{u_7\}, \{u_8\}\}, \\ c_2 &= \{\{u_1, u_5\}, \{u_2, u_6\}, \{u_3, u_7\}, \{u_4, u_8\}\}. \end{aligned}$$

Since the greedy algorithm goes for the largest subset first we will get the cover: $\{\{u_1, u_2, u_3, u_4\}, \{u_5\}, \{u_6\}, \{u_7\}, \{u_8\}\}$ (the size of this cover is 5), whereas the most efficient cover is $c_2$, $(t(\mathcal{N}, \mathcal{R}) = 4)$.

To get around this, the centre should compare the result of greedy algorithm to the cover you would get using each individual tree. Should any one tree give a smaller cover than the output of the algorithm, then it should be used instead. This would require the loop in Table 5.2 being added to the end of the algorithm.

This is a good precaution, and guarantees that the cover produced by the algorithm will be at least as small as the minimum cover in the component schemes. But the algorithm is still not guaranteed to find the smallest cover.

| | |
|---|---|
| 9: | **for** $i$ from 1 to $X$ do |
| 10: | if $\lvert t^{RS_i}(\mathcal{N}, \mathcal{R})\rvert < \lvert S\rvert$ then |
| | assign $S$ to be the cover of $\mathcal{N} \setminus \mathcal{R}$ with $RS_i$ |
| 11: | **end do** |

Table 5.2: Check for Cover Algorithm

The above counterexample can be modified to show this. In the example below, the greedy algorithm results in a cover that is just as small as the minimum of all the trees, and yet is not the smallest possible.

*Example* 47. Suppose $\mathcal{N} \setminus \mathcal{R} = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$, and we have 3 different coverings from 3 different schemes:

$$\begin{aligned} c_1 &= \{\{u_1, u_2, u_3, u_4\}, \{u_5\}, \{u_6\}, \{u_7\}, \{u_8\}\}, \\ c_2 &= \{\{u_1, u_5\}, \{u_2, u_6\}, \{u_3\}, \{u_4\}, \{u_6\}, \{u_8\}\}, \\ c_3 &= \{\{u_3, u_7\}, \{u_4, u_8\}, \{u_1\}, \{u_2\}, \{u_5\}, \{u_6\}\}. \end{aligned}$$

Since the greedy algorithm goes for the largest subset first we will get the cover: $\{\{u_1, u_2, u_3, u_4\}, \{u_5\}, \{u_6\}, \{u_7\}, \{u_8\}\}$ (the size of this cover is 5). This cover is equal in size to the minimum cover of all the component schemes. However, the most efficient cover is $\{\{u_1, u_5\}, \{u_2, u_6\}, \{u_3, u_7\}, \{u_4, u_8\}\}$, $(t(\mathcal{N}, \mathcal{R}) = 4)$.

One final note on the greedy algorithm. The algorithm was constructed specifically so that the output would be a disjoint cover. If this requirement is relaxed, then a less restrictive algorithm can be used. This is done by replacing line 5 with *Find the set $s \in S'$ such that $s \cap P$ has the largest cardinality, add it to $S$*. The modified algorithm could have more sets to choose from when creating the cover $S$, at the very least it will have all those available in the original algorithm. Consequently, the modified algorithm will create a cover at least as small as the original algorithm. Unfortunately, this modification does not prevent outputting non-minimal covers as in the above example.

Even though we showed that it can find smaller covers than the minimal of the component schemes, we cannot guarantee that the greedy will output the minimal cover. In order to be as efficient as possible, the centre should

choose the component schemes carefully to make finding the cover easier. The unions we will be looking at will have a very specific structure. In deriving the formula for $t_{\max}(n, r)$ for the Forest of Trees Schemes we will look at later, we will discover an easy way to form the cover all the privileged users.

### 5.1.3 Disjoint Union

The second way of combining schemes forms a different user set. We will take two schemes whose user sets are non-intersecting and the same size, and create a new scheme whose user set is their union. We will only combine two schemes, unlike the previous method, which could combine many.

*Definition* 48. Let $RS_1 = (\mathcal{N}_1, \Omega_1, \gamma_1), RS_2 = (\mathcal{N}_2, \Omega_2, \gamma_2)$ be two Revocation Schemes with $|\mathcal{N}_1| = |\mathcal{N}_2| = n$ and $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$. We define the *disjoint union* of $RS_1$ and $RS_2$ to be $RS_3 = (\mathcal{N}_3, \Omega_3, \gamma_3)$ where:

$$\begin{aligned}
\mathcal{N}_3 &= \mathcal{N}_1 \cup \mathcal{N}_2 \\
\Omega_3 &= \{(1, j) | j \in \Omega_1\} \cup \{(2, j) | j \in \Omega_2\} \cup \{(3, 0)\} \\
|\Omega_3| &= |\Omega_1| + |\Omega_2| + 1 \\
\gamma_3(i, j) &= \begin{cases} \gamma_1(j) & \text{if } i = 1 \\ \gamma_2(j) & \text{if } i = 2 \\ \mathcal{N}_3 & \text{if } i = 3. \end{cases}
\end{aligned}$$

Again, it is simple to show this results in a Revocation Scheme.

*Lemma* 49. Let $RS_3 = (\mathcal{N}_3, \Omega_3, \gamma_3)$ be the disjoint union of the two Revocation Schemes $RS_1 = (\mathcal{N}_1, \Omega_1, \gamma_1)$ and $RS_2 = (\mathcal{N}_2, \Omega_2, \gamma_2)$. Then $RS_3$ is also a Revocation Scheme.

*Proof.* Since $\forall u \in \mathcal{N}_1$, there exists a $j$ with $\{u\} = \gamma_1(j) = \gamma_3(1, j)$ (by the definition of $\gamma_3$). Similarly, there exists a $j$ with $\{u\} = \gamma_2(j) = \gamma_3(2, j)$, for all $u \in \mathcal{N}_2$. Since $\mathcal{N}_3 = \mathcal{N}_1 \cup \mathcal{N}_2$, all the singletons of $\mathcal{N}_3$ occur in $RS_3$. □

Suppose $RS_1$ and $RS_2$ are Complete Subtree Revocation Schemes with the leaf lists $[u_1, u_2, u_3, u_4]$ and $[u_5, u_6, u_7, u_8]$ respectively. Then the disjoint union of $RS_1$ and $RS_2$ is just a Complete Subtree Revocation Scheme with leaf list $[u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8]$. The index of the root is $(3, 0)$ ($\gamma_3(3, 0) = \mathcal{N}_3$).

The indices of the left half of the tree are all in $RS_1$, and those of the right are in $RS_2$. The property that a complete binary tree has two smaller complete binary trees rooted at the children of the root can also be applied to Complete Subtree Revocation Schemes.

Forming a cover in a disjoint union is not as complicated as it is with a normal union. If $\mathcal{R} = \emptyset$ then use $\gamma_3(3, 0)$, otherwise you form a cover of $\mathcal{N}_1 \setminus \mathcal{R}$ with $RS_1$ and $\mathcal{N}_2 \setminus \mathcal{R}$ with $RS_2$. We can also describe $t_{\max}(n, r)$ for the disjoint union, but first we need to prove the following lemma:

*Lemma* 50. Let $RS_1 = (\mathcal{N}_1, \Omega_1, \gamma_1)$ and $RS_2 = (\mathcal{N}_2, \Omega_2, \gamma_2)$ be two Revocation Schemes, and let $RS = (\mathcal{N}_3, \Omega_3, \gamma_3)$ be their disjoint union. Let $S$ be a cover of $\mathcal{N}_3 \setminus \mathcal{R}$ where $\mathcal{R} \neq \emptyset$. Let $\mathcal{R}_1 = \mathcal{R} \cap \mathcal{N}_1$ and $\mathcal{R}_2 = \mathcal{R} \cap \mathcal{N}_2$. If $S_1 = \{s \in S | s \subseteq \mathcal{N}_1\}$ and $S_2 = \{s \in S | s \subseteq \mathcal{N}_2\}$ then $S$ is a minimal cover of $\mathcal{N}_3 \setminus \mathcal{R}$ if and only if $S_1$ and $S_2$ are minimal covers of $\mathcal{N}_1 \setminus \mathcal{R}_1$ and $\mathcal{N}_2 \setminus \mathcal{R}_2$ respectively.

*Proof.* Firstly, we want to show that if $S$ is a cover of $\mathcal{N}_3 \setminus \mathcal{R}$ then $S_1$ and $S_2$ are covers of $\mathcal{N}_1 \setminus \mathcal{R}_1$ and $\mathcal{N}_2 \setminus \mathcal{R}_2$. Since $\mathcal{N}_1 \cup \mathcal{N}_2 = \mathcal{N}_3$ and $\mathcal{R} \subseteq \mathcal{N}$, we have:

$$\begin{aligned}(\mathcal{N}_1 \setminus \mathcal{R}_1) \cup (\mathcal{N}_2 \setminus \mathcal{R}_2) &= (\mathcal{N}_1 \cup \mathcal{N}_2) \setminus (\mathcal{R}_1 \cup \mathcal{R}_2) \\ &= \mathcal{N}_3 \setminus ((\mathcal{R} \cap \mathcal{N}_1) \cup (\mathcal{R} \cap \mathcal{N}_2)) \\ &= \mathcal{N}_3 \setminus \mathcal{R}.\end{aligned}$$

Since $\mathcal{R} \neq \emptyset$, $(\mathcal{N}_3 \setminus \mathcal{R}) \neq \mathcal{N}_3$. This means that any set $s \in S$ is strictly contained in $\mathcal{N}_3$. By the definition of a disjoint union, if $s \neq \mathcal{N}_3$ then $s \subseteq \mathcal{N}_1$ or $s \subseteq \mathcal{N}_2$. So we have:

$$S_1 \cup S_2 = \{s \in S | s \subseteq \mathcal{N}_1\} \cup \{s \in S | s \subseteq \mathcal{N}_2\} = \{s \in S | s \subseteq \mathcal{N}_3\} = S.$$

So if $S_1$ is a cover of $\mathcal{N}_1 \setminus \mathcal{R}_1$ and $S_2$ is a cover of $\mathcal{N}_2 \setminus \mathcal{R}_2$ then $S$ is a cover of $\mathcal{N}_3 \setminus \mathcal{R}$, and vice versa, since they cover the exact same set of users. We now have to show the minimality condition crosses over.

Assume that $S_1$ and $S_2$ are minimal covers of $\mathcal{N}_1 \setminus \mathcal{R}_1$ and $\mathcal{N}_2 \setminus \mathcal{R}_2$ respectively. Suppose that their union $S$ is not the minimum cover of $\mathcal{N}_3 \setminus \mathcal{R}$, and that their exists a smaller cover $S'$. $S'$ can be partitioned into $\{s \in S' | s \subseteq \mathcal{N}_1\}$ and $\{s \in S' | s \subseteq \mathcal{N}_2\}$, just like $S_1$ and $S_2$ were. Since the sum of the sizes of these sets add up to $|S'| < |S|$ and $|S| = |S_1| + |S_2|$, at least one of these sets

is smaller than its counterpart ($S_1$ or $S_2$). This contradicts the assumption that $S_1$ and $S_2$ are minimal covers. So if $S_1$ and $S_2$ are minimal covers, $S$ is a minimal cover.

Similarly, assume that $S$ is a minimal cover of $\mathcal{N}_3 \setminus \mathcal{R}$. Suppose that one of $S_1$ and $S_2$ (the partition of $S$ over $\mathcal{N}_1$ and $\mathcal{N}_2$) are not minimal covers. Without loss of generality, say there is a cover $S_1'$ of $\mathcal{N}_1 \setminus \mathcal{R}_1$ with $|S_1'| < |S_1|$. If we take the union of $S_1'$ and $S_2$ we will get a cover of $\mathcal{N}_3 \setminus \mathcal{R}$. But since $S_1'$ and $S_2$ are disjoint:

$$|S_1' \cup S_2| = |S_1'| + |S_2| < |S_1| + |S_2| = |S|.$$

Therefore, we get a cover smaller than $S$. This contradicts our assumption. So if $S$ is a minimal cover then $S_1$ and $S_2$ are both minimal covers. □

*Corollary* 51. Let $RS_1 = (\mathcal{N}_1, \Omega_1, \gamma_1), RS_2 = (\mathcal{N}_2, \Omega_2, \gamma_2)$ be two Revocation Schemes, and let $RS = (\mathcal{N}_3, \Omega_3, \gamma_3)$ be their disjoint union. Then provided $\mathcal{R} \neq \emptyset$:

$$t^{RS}(\mathcal{N}_3, \mathcal{R}) = t^{RS_1}(\mathcal{N}_1, \mathcal{R}_1) + t^{RS_2}(\mathcal{N}_2, \mathcal{R}_2).$$

*Proof.* This follows immediately since $t(\mathcal{N}, \mathcal{R})$ is just the size of the minimal cover. □

Using this, we can state a formula for $t_{\max}(n, r)$ of a disjoint union Revocation scheme.

*Theorem* 52. Let $RS_1 = (\mathcal{N}_1, \Omega_1, \gamma_1), RS_2 = (\mathcal{N}_2, \Omega_2, \gamma_2)$ be two Revocation Schemes, and let $RS = (\mathcal{N}_3, \Omega_3, \gamma_3)$ be their disjoint union. Then:

$$\begin{aligned} t^{RS}_{\max}(2n, 0) &= 1 \qquad (5.2) \\ t^{RS}_{\max}(2n, r) &= \max_{\substack{0 \le r_1, r_2 \le \min(r,n) \\ r_1 + r_2 = r}} \left( t^{RS_1}_{\max}(n, r_1) + t^{RS_2}_{\max}(n, r_2) \right), \qquad \text{for all } 1 \le r \le 2n. \end{aligned}$$

*Proof.* When $r = 0$, $\gamma_3(3, 0) = \mathcal{N}_3$ gives a cover of size 1, which means $t_{\max}(n, r)$ must be at most 1. Any cover of a non-empty set of users must be non-empty, so in this case $t_{\max}(n, r)$ must be at least one. Therefore $t^{RS}_{\max}(2n, 0) = 1$.

Otherwise, $r \ge 0$. Let

$$t' = \max_{\substack{0 \le r_1, r_2 \le \min(r,n) \\ r_1 + r_2 = r}} \left( t^{RS_1}_{\max}(n, r_1) + t^{RS_2}_{\max}(n, r_2) \right).$$

Let $S_1$ be a cover of $\mathcal{N}_1 \setminus \mathcal{R}_1$ of size $t^{RS_1}_{\max}(n, r_1)$, where $r_1$ provides the maximum in the above expression. Let $S_2$ be a cover of $\mathcal{N}_2 \setminus \mathcal{R}_2$ of size $t^{RS_2}_{\max}(n, r_2)$, for $r_2 = r - r_1$. So $|S_1| + |S_2| = t'$. In Lemma 50 we showed that because $S_1$ and $S_2$ are minimal covers, $S$ is a minimal cover of

$$(\mathcal{N}_1 \setminus \mathcal{R}_1) \cup (\mathcal{N}_2 \setminus \mathcal{R}_2) = \mathcal{N}_3 \setminus \mathcal{R}.$$

Therefore $t^{RS}(\mathcal{N}_3, \mathcal{R}) = |S| = |S_1| + |S_2| = t'$. But because $t^{RS}(\mathcal{N}_3, \mathcal{R}) \leq t^{RS}_{\max}(2n, r)$, that means:

$$t^{RS}_{\max}(2n, r) \geq t'. \tag{5.3}$$

Let $\mathcal{R}$ be a non-empty subset such that $t^{RS}(\mathcal{N}_3, \mathcal{R}) = t^{RS}_{\max}(2n, r)$. If we put $\mathcal{R}_1 = \mathcal{R} \cap \mathcal{N}_1$ and $\mathcal{R}_2 = \mathcal{R} \cap \mathcal{N}_2$, then by Corollary 51 we have:

$$\begin{aligned}
t^{RS}_{\max}(2n, r) &= t^{RS}(\mathcal{N}_3, \mathcal{R}) \\
&= t^{RS_1}(\mathcal{N}_1, \mathcal{R}_1) + t^{RS_2}(\mathcal{N}_2, \mathcal{R}_2) \\
&\leq t^{RS_1}_{\max}(n, |\mathcal{R}_1|) + t^{RS_2}_{\max}(n, |\mathcal{R}_2|) \\
&\leq \max_{\substack{0 \leq r_1, r_2 \leq \min(r,n) \\ r_1 + r_2 = r}} \left( t^{RS_1}_{\max}(n, r_1) + t^{RS_2}_{\max}(n, r_2) \right) \\
&= t'.
\end{aligned}$$

Combining this with Formula (5.3), we get $t^{RS}_{\max}(2n, r) = t'$. □

There is a similar formula for $t_{aver}(n, r)$, but first we need the following result. In Chapter 4 we found a way to express the set $\{s : s \in \mathcal{N} \ \ |s| = r\}$ ($|\mathcal{N}| = n$) in terms of subsets from a smaller set $|\mathcal{N}'| = n/2$. From this we were able to work out the recursive relation of $t_{aver}(n, r)$ for the Complete Subtree Revocation Scheme. For a disjoint union, we need to write all subsets of $\mathcal{N}_3$ of size $r$ in terms of unions of subsets from $\mathcal{N}_1$ and $\mathcal{N}_2$. As required for a disjoint union, the latter two sets have to be the same size.

*Lemma* 53. Let $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3$ be three sets with the following properties: $\mathcal{N}_3 = \mathcal{N}_1 \cup \mathcal{N}_2$, $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$ and $|\mathcal{N}_1| = |\mathcal{N}_2| = n$. Then:

$$\begin{aligned}
\{s \subseteq \mathcal{N}_3 : |\mathcal{R}| = r\} = \bigcup_{r_1 = \max(0, r-n)}^{\min(n,r)} \{\mathcal{R}_1 \cup \mathcal{R}_2 : &\mathcal{R}_1 \subseteq \mathcal{N}_1, \ |\mathcal{R}_1| = r_1, \\
&\mathcal{R}_2 \subseteq \mathcal{N}_2, \ |\mathcal{R}_2| = r - r_1\}.
\end{aligned} \tag{5.4}$$

*Proof.* Let $LHS$ be the left-hand side of Formula (5.4), and $RHS$ be the right-hand side of Formula (5.4). Consider any two subsets $\mathcal{R}_1$ and $\mathcal{R}_2$ from $RHS$. $|\mathcal{R}_1| = r_1$, for some $r_1$ in the range $[\max(0, r-n), \ldots, \min(n, r)]$. Since $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$ and $\mathcal{R}_1 \subseteq \mathcal{N}_1,\ \mathcal{R}_2 \subseteq \mathcal{N}_2$, we have that $\mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset$. Therefore:

$$|\mathcal{R}_1 \cup \mathcal{R}_2| = |\mathcal{R}_1| + |\mathcal{R}_2| = r_1 + (r - r_1) = r.$$

This subset is contained in $\mathcal{N}_3$ as $\mathcal{R}_1 \cup \mathcal{R}_2 \subseteq \mathcal{N}_1 \cup \mathcal{N}_2 = \mathcal{N}_3$. So $\mathcal{R}_1 \cup \mathcal{R}_2 \in LHS$, for all $\mathcal{R}_1 \cup \mathcal{R}_2 \in RHS$, which means $LHS \supseteq RHS$.

Consider any set $\mathcal{R}$ from $LHS$ ($\mathcal{R} \subseteq \mathcal{N}_3,\ |\mathcal{R}| = r$). This set can be partitioned into two non-overlapping subsets $\mathcal{R} \cap \mathcal{N}_1$ and $\mathcal{R} \cap \mathcal{N}_2$, since $\mathcal{R} \subseteq \mathcal{N}_3$ and $\mathcal{N}_3 = \mathcal{N}_1 \cup \mathcal{N}_2$ (which are disjoint). We know that $|\mathcal{R} \cap \mathcal{N}_1|$ is non-negative and less than or equal to $\min(n, r)$ (since $|\mathcal{N}_1| = n$ and $|\mathcal{R}| = r$). The same is true for $|\mathcal{R} \cap \mathcal{N}_2|$, and also:

$$\begin{aligned} |\mathcal{R} \cap \mathcal{N}_1| + |\mathcal{R} \cap \mathcal{N}_2| &= r \\ |\mathcal{R} \cap \mathcal{N}_1| &= r - |\mathcal{R} \cap \mathcal{N}_2|. \end{aligned}$$

Because $|\mathcal{R} \cap \mathcal{N}_2| \leq n$ that implies $|\mathcal{R} \cap \mathcal{N}_1| \geq r - n$. So we have that $\mathcal{R}$ is the union of two subsets $\mathcal{R} \cap \mathcal{N}_1 \subseteq \mathcal{N}_1$ and $\mathcal{R} \cap \mathcal{N}_2 \subseteq \mathcal{N}_2$, with $|\mathcal{R} \cap \mathcal{N}_1|$ in the range $[\max(0, r-n), \ldots, \min(n, r)]$ and $|\mathcal{R} \cap \mathcal{N}_2| = r - |\mathcal{R} \cap \mathcal{N}_1|$. But these sets have all the properties of sets from $RHS$. Therefore $\mathcal{R} \in RHS$ for all $\mathcal{R} \in LHS$, which means $LHS \subseteq RHS$. Coupled with the above, this proves $LHS = RHS$. □

We now have all the results needed to prove the recursive relation for $t_{aver}(n, r)$ for any disjoint union of two schemes. The proof of the relation is just a matter of combining Corollary 51 and Lemma 53 with the definition of $t_{aver}(n, r)$.

*Corollary* 54. Let $RS_1 = (\mathcal{N}_1, \Omega_1, \gamma_1), RS_2 = (\mathcal{N}_2, \Omega_2, \gamma_2)$ be two Revocation Schemes where $|\mathcal{N}_1| = |\mathcal{N}_2| = n$. Let $RS = (\mathcal{N}_3, \Omega_3, \gamma_3)$ be their disjoint union ($|\mathcal{N}_3| = 2n$). Then $t^{RS}_{aver}(2n, 0) = 1$ and:

$$t^{RS}_{aver}(2n, r) = \sum_{r_1 = \max(0, r-n)}^{\min(n,r)} \frac{\binom{n}{r_1}\binom{n}{r-r_1}\left(t^{RS_1}_{aver}(n, r_1) + t^{RS_2}_{aver}(n, r - r_1)\right)}{\binom{2n}{r}}. \quad (5.5)$$

for all $1 \leq r \leq 2n$.

*Proof.* The definition of $t_{aver}(n,r)$ is:

$$t^{RS}_{aver}(2n,r) = \sum_{\substack{\mathcal{R}\subseteq\mathcal{N}_3\\|\mathcal{R}|=r}} \frac{t^{RS}(\mathcal{N}_3,\mathcal{R})}{\binom{2n}{r}}.$$

If $r=0$, then there is only one possibility for $\mathcal{R}=\emptyset$. As the construction of a disjoint union has an index $\gamma(3,0)=\mathcal{N}_3$, we have $t_{aver}(2n,0)=t^{RS}(\mathcal{N}_3,\emptyset)=1$.

Lemma 53 gives us a way to sum over the subsets of $\mathcal{N}_3$ with subsets from $\mathcal{N}_1$ and $\mathcal{N}_2$. And by Corollary 51, $t^{RS}(\mathcal{N}_3,\mathcal{R})$ is equal to the sum of the minimal covers in $\mathcal{N}_1$ and $\mathcal{N}_2$. The corollary is applicable in this case because $r\neq 0$, which means $\mathcal{R}\neq\emptyset$. Combining these two we get:

$$t^{RS}_{aver}(2n,r) = \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \sum_{\substack{\mathcal{R}_1\subseteq\mathcal{N}_1\\|\mathcal{R}_1|=r_1}} \sum_{\substack{\mathcal{R}_2\subseteq\mathcal{N}_2\\|\mathcal{R}_1|=r-r_1}} \frac{(t^{RS_1}(\mathcal{N}_1,\mathcal{R}_1)+t^{RS_2}(\mathcal{N}_2,\mathcal{R}_2))}{\binom{2n}{r}}.$$

where $\mathcal{R}_1=\mathcal{R}\cap\mathcal{N}_1$ and $\mathcal{R}_2=\mathcal{R}\cap\mathcal{N}_2$. A little rearranging of the terms, and applying the definition of $t_{aver}(n,r)$ and we get:

$$\begin{aligned}
t^{RS}_{aver}(2n,r) = \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \frac{1}{\binom{2n}{r}}\Bigg( &\sum_{\substack{\mathcal{R}_1\subseteq\mathcal{N}_1\\|\mathcal{R}_1|=r_1}} \sum_{\substack{\mathcal{R}_2\subseteq\mathcal{N}_2\\|\mathcal{R}_1|=r-r_1}} t^{RS_1}(\mathcal{N}_1,\mathcal{R}_1)\\
&+ \sum_{\substack{\mathcal{R}_1\subseteq\mathcal{N}_1\\|\mathcal{R}_1|=r_1}} \sum_{\substack{\mathcal{R}_2\subseteq\mathcal{N}_2\\|\mathcal{R}_1|=r-r_1}} t^{RS_2}(\mathcal{N}_2,\mathcal{R}_2)\Bigg)\\
t^{RS}_{aver}(2n,r) = \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \frac{1}{\binom{2n}{r}}\Bigg( &\binom{n}{r-r_1} \sum_{\substack{\mathcal{R}_1\subseteq\mathcal{N}_1\\|\mathcal{R}_1|=r_1}} t^{RS_1}(\mathcal{N}_1,\mathcal{R}_1)\\
&+ \binom{n}{r_1} \sum_{\substack{\mathcal{R}_2\subseteq\mathcal{N}_2\\|\mathcal{R}_1|=r-r_1}} t^{RS_2}(\mathcal{N}_2,\mathcal{R}_2)\Bigg)\\
t^{RS}_{aver}(2n,r) = \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \frac{1}{\binom{2n}{r}}\Bigg( &\binom{n}{r-r_1}\binom{n}{r_1} t^{RS_1}_{aver}(n,r_1)\\
&+ \binom{n}{r_1}\binom{n}{r-r_1} t^{RS_2}_{aver}(n,r-r_1)\Bigg)\\
t^{RS}_{aver}(2n,r) = \sum_{r_1=\max(0,r-n)}^{\min(n,r)} &\frac{\binom{n}{r_1}\binom{n}{r-r_1} t^{RS_1}_{aver}(n,r_1) + t^{RS_2}_{aver}(n,r-r_1)}{\binom{2n}{r}}.
\end{aligned}$$

□

Note that it is possible to form a disjoint union of a scheme with itself, provided you relabel the users in one of the schemes. The size of the users set doubles, and you get all the subsets relating to the original scheme on both halves of the users set, plus one index for the entire user set. This is exactly what happens when we double the parameter $n$ in the Complete Subtree Revocation Scheme.

The scheme with $2n$ users has its function $\gamma$ defined by a complete binary tree with $2n$ leaves. Earlier we showed that such a tree can be divided into two complete binary trees, each with $n$ leaves, with one node leftover. This works the other way around as well, building a tree with $2n$ leaves from two trees with $n$ leaves. Additionally, this is done in exactly the same way as we form a disjoint union. Forming a copy of the user set corresponds to adding another tree with different leaves, adding an index $i$ such that $\gamma(i)$ equals the combined user sets corresponds to the adding a node to be the root connecting the two subtrees. This is why we can make the above claim. It also gives us another recursive formula for $t_{aver}^{CSRS}(n, r)$ (we already know $t_{\max}^{CSRS}(n, r)$ explicitly). The difference in the two formulae comes from the way the recursion algorithm "works back" to the smaller scheme. In the earlier relation (Formula (4.8)), we reduce the scheme with $2n$ users to a single scheme with $n$ users by removing the leaves. The relation in this chapter reduces the larger $2n$ scheme to two schemes with $n$ users by removing the root. These results will be very helpful for analysing other tree-based schemes.

The constructions presented are general, but some inferences can be made. The first union is a way to combine two schemes and get the shortest bandwidth. If one scheme has the lower bandwidth for low values of $r$, and the other has lower bandwidth for high values of $r$, then the union will have the lower bandwidth of the two in each range. It is impossible to calculate the storage of the union scheme without details of the components, but we know it is at least slightly lower than the combined storage of the components. At the very least, the singletons are common to both schemes, so there is at least one key common to both schemes that does not have to be stored twice. The disjoint union is a natural way to "grow" a scheme to twice the population size. We have a formula for $t_{\max}(n, r)$, but as it depends on $t_{\max}(n, r)$ of the

component schemes, we cannot make any comment on the bandwidth in general. The storage of a disjoint union does have the nice property that it will only ever increase by 1 as $n$ is doubled. The number of keys any user will need to store is just those that were stored in the component scheme plus the key shared by the combined user sets. In the next two sections, we will show how to use these constructions to create efficient schemes.

## 5.2 Complete Forest

The first scheme to be built using the above constructions is the Complete Forest Revocation Scheme. Before describing it, we will explain the motivation. A good place to start looking at combining different schemes is to consider what causes the Complete Subtree Revocation Scheme to have high $t_{\max}(n, r)$. To get $t(\mathcal{N}, \mathcal{R}) = n/2$, we simply revoke every second user. This causes $t(\mathcal{N}, \mathcal{R})$ to be high because no subset of two or more privileged users share a key and so all keys for the transmission are those at the leaf level. However, if we have a well-chosen union of Complete Subtree Revocation Schemes, then hopefully users that are not siblings in one scheme are siblings in at least one other. Remember that two leaves are siblings if they have the same parent, so two sibling users will be the only users who have the key associated with their parent. So we can define such a scheme as follows:

*Definition* 55. Let $RS = (\mathcal{N}, \Omega', \gamma')$ be the union of $X$ Complete Subtree Revocation Schemes, $RS_1 = (\mathcal{N}, \Omega, \gamma_{f_1}), \ldots, RS_X = (\mathcal{N}, \Omega, \gamma_{f_X})$. We say that $RS$ is a *Complete Forest Revocation Scheme* if for all pairs of users $u_{i_1}, u_{i_2} \in \mathcal{N}$ there exists an index $i \in \Omega'$ such that:

$$\gamma_{f'}(i) = \{u_{i_1}, u_{i_2}\}.$$

Requiring every pair of users to be siblings (i.e. having the same parent, or next highest node) in at least one of the trees will require taking the union of several schemes. It is possible to work out exactly how many schemes are needed. There are $n/2$ pairs of leaves in any one tree, and a total of $\binom{n}{2} = \frac{n(n-1)}{2}$ different pairs. So assuming that there is no repetition of pairs, we would need $\frac{n(n-1)}{2} / \frac{n}{2} = n - 1$ trees. No repetition means that every pair of

leaves are siblings in exactly one tree. We will now describe how to construct a family of trees with this property, starting from the simplest case, and building the trees iteratively. Since we will only be dealing with unions of the Complete Subtree Revocation Scheme, we will use the leaf list as representation. The first scheme in the union will always have $\gamma_f$, giving the leaf list $[u_1, u_2, \ldots, u_n]$.

For the smallest binary tree, $n = 2$, the condition holds with just the one tree, $[u_1, u_2]$. The only pair of nodes that can be picked are $\{u_1, u_2\}$ and they are siblings. We can better illustrate the general construction by showing how we go from $n = 2$ to $n = 4$. Naturally, the first tree is $[u_1, u_2, u_3, u_4]$. We now just need 2 trees to pair all leaves from the first half $(u_1, u_2)$ with those in the second $(u_3, u_4)$: $[u_1, u_3, u_2, u_4], [u_1, u_4, u_2, u_3]$. All $\binom{4}{2} = 6$ pairs occur as siblings in one of these three trees. Also, three is the minimal number of trees since there are 6 pairs of siblings needed and only 2 pairs of siblings per tree. Note that when $n = 2$, we needed 1 $(= n - 1)$ tree and when $n = 4$ we needed 3 $(= n - 1)$, in keeping with what we said earlier. We will describe a construction of $n - 1$ trees for any $n$ (a power of two), and show that it is a Complete Forest Revocation Scheme. As the construction uses modular arithmetic, we will say that $(i \mod n)$ returns the least positive residue, i.e. an integer in the range $1, \ldots, n$.

### 5.2.1 Complete Forest Construction Algorithm

Assume that we have $RS^1 = (\mathcal{N}^1, \Omega^1, \gamma^1)$, a Complete Forest Revocation Scheme with $n$ users, comprised of the union of $n - 1$ Complete Subtree Revocation Schemes. We will now construct $RS'$, a Complete Forest Revocation Scheme with $2n$ users. We need to form a disjoint union of $RS^1$ with $RS^1$ defined on a different user set. By default, $\mathcal{N}^1 = \{u_1, u_2, \ldots, u_n\}$, so define $\mathcal{N}^2 = \{u_{n+1}, u_{n+2}, \ldots, u_{2n}\}$. Let $\gamma^2 : \Omega^2 \to 2^{\mathcal{N}^2}$ be a function isomorphic to $\gamma^1$. So if we put $RS^2 = (\mathcal{N}^2, \Omega^2, \gamma^2)$, then we get a Revocation Scheme identical to $RS^1$, but with a completely different user set (note that $\mathcal{N}^1 \cap \mathcal{N}^2 = \emptyset$). Let $RS^3 = (\mathcal{N}^1 \cup \mathcal{N}^2, \Omega^3, \gamma^3)$ be the disjoint union of $RS^1$ and $RS^2$. $RS^3$ can also be considered the union of $n - 1$ Complete Subtree Revocation Schemes on $2n$ users, as we have already shown that the $2n$ user scheme is the same as getting the disjoint union of the $n$ user scheme with a copy of itself. As

$$
\begin{array}{rcccccccc}
\text{Tree}_n & = & [u_1, & u_{n+1}, & u_2, & u_{n+2}, & \ldots, & u_n, & u_{2n} & ], \\
\text{Tree}_{n+1} & = & [u_1, & u_{n+2}, & u_2, & u_{n+3}, & \ldots, & u_n, & u_{n+1} & ], \\
\vdots & = & & \vdots & & & \ddots & & & \\
\text{Tree}_{2n-1} & = & [u_1, & u_{2n}, & u_2, & u_{n+1}, & \ldots, & u_n, & u_{n+n-1} & ].
\end{array}
$$

Table 5.3: Leaf list for trees in Complete Forest Revocation Scheme

it is, $RS^3$ is not enough for a Complete Forest on $2n$ users (we need at least $2n-1$ schemes). The fact that we have two copies of a Complete Forest with $n$ users only guarantees that any pair of users $u_{i_1}, u_{i_2}$ that are either both picked from $\mathcal{N}^1$, or both picked from $\mathcal{N}^2$, will correspond to an index in one of the schemes, $\gamma^3(j) = \{u_{i_1}, u_{i_2}\}$, for some $j \in \Omega^3$.

What we need to do now is exactly the same as what we did in the $n = 4$ case. The user set is divided into two halves: $\mathcal{N}^1 = \{u_1, \ldots, u_n\}$ and $\mathcal{N}^2 = \{u_{n+1}, \ldots, u_{2n}\}$. We need to create schemes such that every possible pair of users, one from each half, are siblings in one scheme. We then have to form the union of these with $RS^3$. There are no such siblings in $RS^3$, since in that scheme all siblings are both from the same half. That means all $n^2$ (size of each half is $n$) possibilities must be accounted for. Without repetitions, this will take exactly $n$ trees: $n^2$ pairs of siblings needed, $n$ pairs per tree (trees are length $2n$, which means $n^2/n = n$ trees). One way of doing this is to use schemes with the leaf lists in Table 5.3.

We can define these more explicitly as:

$$
\begin{gathered}
\text{Tree}_k = [a_1, a_2, \ldots, a_{2n}], k = n, \ldots, 2n-1, \\
a_{2i-1} = u_i,\ a_{2i} = u_j,\ \text{where } j = (i + k \mod n) + n,\ i = 1, \ldots, n. \quad (5.6)
\end{gathered}
$$

What this is saying is that $2n$ leaves are assigned users in the different trees as follows:

The users from $\mathcal{N}^1$ are assigned to the odd leaves ($1^{st}, 3^{rd}, 5^{th}, \ldots$), in the same way for all trees: User $u_i$ is assigned leaf $v_{2n-1+2i}$. The users from $\mathcal{N}^2$ are assigned to the even leaves ($2^{nd}, 4^{th}, 6^{th}, \ldots$), but in a manner that differs for each tree. In $tree_k$, $u_i$ is assigned leaf $v_{2n-2+j}$ where $j = (i + k \mod n) + n$. This gives a cyclic shift to the left to all the users in $\mathcal{N}^2$ when going from one

tree to the next. The Complete Forest is then the union of these $n$ schemes with the $n-1$ schemes in $RS^3$, hence the labelling $n, \ldots, 2n-1$.

We will use induction to prove that these trees satisfy the required condition.

*Lemma* 56. Let $RS = (\mathcal{N}, \Omega', \gamma')$ be the output of the algorithm described in Section 5.2.1 after $k-1$ iterations. Then $|\mathcal{N}| = 2^k$, and for all pairs of users $u_{i_1}, u_{i_2} \in \mathcal{N}$ there exists an index $i \in \Omega'$ such that:

$$\gamma_{f'}(i) = \{u_{i_1}, u_{i_2}\}.$$

*Proof.* The user set is doubled in size each time the construction is run. Since we start with the leaf list $[u_1, u_2]$, of size 2, each iteration multiplies the length by 2. So $n$ is just 2 times 2 to the power of the number of iterations:

$$n = 2.2^{k-1} = 2^k.$$

Let the induction hypothesis be that $k-1$ iterations of the construction create a Complete Forest Revocation Scheme (which we know has $n$ users). The initial cases of $n = 2$ and $n = 4$ have already been proven. We assume the result is true for $n$, and use the above construction to show that it is true for $2n$. So we have a union of $2n-1$ Complete Subtree Revocations Schemes, constructed as outlined in Section 5.2.1. The first $n-1$ schemes guarantee that any pair of users chosen are siblings in one tree provided that they are chosen from the same half (by the induction hypothesis this is the disjoint union of two Complete Forest Revocation Schemes).

So it remains to show that pairs of users from different halves are siblings in some tree. Without loss of generality we will consider two users $(u_i, u_j)$ where $i$ is in the range $1 \leq i \leq n$ and $j$ is in the range $n+1 \leq j \leq 2n$. Another way to write $j$ is $j = j' + n$, where $1 \leq j' \leq n$. Because siblings in the first $n-1$ trees are only from the same half, we know we need to look for this pair in the second lot of $n$ trees. We know that $u_i$ will always be in the $(2i-1)^{th}$ column of any of these trees, we just need to find which tree has $u_i$ paired with $u_j$. But we can just use Formula (5.6) to find out which tree we

need:

$$\begin{aligned} j' + n &= ((i+k) \mod n) + n \\ j' &= (i+k) \mod n \\ k &\equiv j' - i \mod n. \end{aligned}$$

So $u_i$ and $u_j$ are siblings in $\text{Tree}_k$, where $k = j' - i \mod n$, since $j = ((i+k) \mod n) + n$, which is what the sibling of $i$ in this tree is defined to be. □

This is how we construct a Complete Forest Revocation Scheme. This is not the only way to construct a Forest of $n-1$ Complete Subtree Revocation Schemes such that every pair of users are siblings in one tree. There are many other ways of finding similar forests. It is equivalent to decomposing the complete graph on $n$ vertices into $n-1$ subgraphs of $n/2$ disjoint edges. The $n$ vertices are the $n$ users. Each edge joins two vertices and represents a pair of siblings. Since each tree has $n/2$ pairs of siblings, $n/2$ edges is equivalent to a tree. The edges need to be disjoint as each user is only assigned one leaf in any tree. The method above is very simply described, and easy to analyze, as well as having some desirable qualities when it comes to storage.

### 5.2.2 Storage

What can we say about the performance of this Complete Forest Revocation Scheme? Obviously the amount of storage needed has increased when compared to the Complete Subtree Revocation Scheme. It appears to have been multiplied by a factor of $n-1$, as it is comprised of $n-1$ copies of this scheme. This suggests $|U|_{\max} = (k+1)(n-1) = (k+1)(2^k - 1)$. However, we have already shown that there are subsets common to different trees, regardless of how they are chosen. Namely, the singletons, and the complete set $\mathcal{N}$. Each user does not need to retain $n-1$ keys that serve the same purpose, i.e. have the same subset of users that share it. These are not the only subsets common to the different trees. We can work out the exact value of the storage, but only for the specific construction of the Complete Forest we have described. Any other construction will not have the properties we will be making use of.

*Lemma* 57. Let $RS$ be a Complete Forest Revocation Scheme on $n = 2^k$ users, constructed as we have outlined in Section 5.2.1. Then $RS$ has:

$$|U|_{\max} = 2^k(k-2) + k + 3.$$

*Proof.* The algorithm to create the $n-1$ Complete Subtree Revocation Schemes works in an iterative manner. We initialise the scheme with the Complete Subtree Revocation Scheme with two users, the leaf list is just $[u_1, u_2]$. At the start of every subsequent step, each existing scheme is replaced with a disjoint union with itself. We also add new schemes at each step, $2^i$ in step $i$. There are $k-1$ steps after initialisation needed to create the union of $n-1$ schemes ($\sum_{i=0}^{k-1} 2^i = 2^k - 1$). In order to work out the storage, we are going to find a general expression for the distinct subsets in the additional schemes at each iteration.

As we have said, the first tree is $[u_1, u_2]$. Consider what happens to this tree during the construction. In each iteration we form a disjoint union of the scheme with itself. In terms of the leaf list, it goes from $[u_1, \ldots, u_a]$ to $[u_1, \ldots, u_{2a}]$. The number of users is doubled each time and since it starts at 2, after step $i$ we have $2^{i+1}$ users. After $k-1$ steps, we get $2.2^{k-1} = 2^k = n$ users. This will just be the Complete Subtree Revocation Scheme with $\gamma_f$ where $f(v_i) = i - (n-1)$. This on its own has a storage requirement of $k+1$ for each user. To determine the storage for the Complete Forest Revocation Scheme, we will compare all trees added by the algorithm to this original tree and see where they differ (more keys stored by any user) and where they are the same (no extra keys). But, when we add trees, they are not of length $n$ (except for the last step) so we can not compare the trees as they are. Fortunately, all trees undergo the same "disjoint union with itself" process until they are all length $n$.

Say we are at the $i^{th}$ iteration of the construction. We will already have a forest of $1 + 2 + 2^2 + \ldots + 2^{i-1} = 2^i - 1$ trees, each of length $2^i$. These trees have the property that any pair of users from $u_1, \ldots, u_{2^i}$ will be siblings in one of the trees. Forming the disjoint union with itself allows the same to be said for any pair of users in $u_{2^i}, \ldots, u_{2^{i+1}}$. The trees that we will add will consist of pairs of siblings, one from each list. The formula for the leaf list of tree $j$

(where $2^i \leq j \leq 2^{i+1} - 1$) is:

$$[u_1, u_{j+1}, u_2, u_{j+2}, \ldots, u_{2^i}, u_{j+2^i}].$$

This leaf list can be considered a sequence of pairs in the form:

$$[u_m, u_l : m \in S_1 = \{1, 2, \ldots, 2^i\}, l \in S_2 = \{2^i + 1, 2^i + 2, \ldots, 2^{i+1} - 1\}],$$

where every second term is reduced by the appropriate modulus. The first $2^i - 1$ trees are (extensions of) Complete Forests of length $2^i$. Each set of size $\leq 2^i$ only has users from either $S_1$ or $S_2$. The only sets with users from both are sets of size $2^{i+1}$ or greater. Since the singletons, $\{u_i\}$, are always common to all trees, for each tree added in the $i^{th}$ iteration, each user will need to store 1 new key for each subset of size $2, 2^2, \ldots, 2^i$. This works out to be $i$ new keys, for the $i$ subsets that do not occur in the previous trees. Since $2^i$ trees are added in the $i^{th}$ iteration, and there are $k - 1$ iterations, the total storage is:

$$\begin{aligned} |U|_{\max} &= k + 1 + \sum_{i=1}^{k-1} i2^i \\ &= k + 1 + 2^k(k-2) + 2 \\ &= 2^k(k-2) + k + 3. \end{aligned}$$ □

Although this is less than the original estimate of $(k+1)(n-1)$, it is still the same order of complexity, $\mathcal{O}(n \log(n))$. However, the distinction of the amount of storage needed will be very important when we describe a variation of the Complete Forest.

### 5.2.3 Bandwidth

In this section, we will find some limits $t_{\max}(n, r)$ for a Complete Forest Revocation Scheme. Some of the bounds arise directly from the union of schemes (Formula (5.1) and Formula (5.2)), but we also use the specific property of a Complete Forest (any pair of users are siblings in at least one tree).

The formula for $t_{\max}(n, r)$ for the Complete Subtree Revocation Scheme is an upper bound for $t_{\max}(n, r)$ in the Complete Forest Revocation Scheme, since all subsets from one Complete Subtree are contained within it. They will both be the same for some values of $r$.

*Lemma* 58. Let $CFRS$ be a Complete Forest Revocation Scheme on $n = 2^k$ users, constructed as we have outlined in Section 5.2.1. Then the following are true for $CFRS$:

$$\begin{array}{llll} 1. & t_{\max}(n,0) = 1, & 2. & t_{\max}(n,n) = 0, \\ 3. & t_{\max}(n,1) = \log_2(n), & 4. & t_{\max}(n,n-1) = 1, \\ 5. & t_{\max}(n,2) = \log_2(n) - 1, & 6. & t_{\max}(n,n-2) = 1. \end{array}$$

*Proof.* The first formula is true, as we can cover the whole user set with the index for the root in any of the $n-1$ Complete Subtree Schemes. Formula 2 is true for all schemes, if there are no privileged users then there is no broadcast.

It is certainly true that $t_{\max}(n,1) \leq \log_2(n)$, since a cover of this size can be made in any of the component schemes. We cannot do any better in the union of all these schemes. The reason for this can be seen when we use a disjoint cover. The number of privileged users is $n-1$, where $n$ is a power of 2. The only subsets that can be used in the cover have cardinality a power of 2. Since the cover must be disjoint, that means the sum of the sizes of the subsets in the cover must equal $2^k - 1$ exactly. The largest subset that could be used in a cover is of size $2^{k-1}$. This would leave $2^{k-1} - 1$ users to cover. The largest subset that we could use in a cover of these remaining users would be of size $2^{k-2}$, leaving $2^{k-2} - 1$. This would continue until we had one subset in the cover for every power of 2 from $k-1$ to 0, $\sum_{i=0}^{k-1} 2^i = 2^k - 1$. This means that any cover must have at least $k = \log_2(n)$ subsets.

We can prove the same result even if we allow overlapping subsets in the cover. Suppose we want to cover $2^k - 1$ users. First, we add to the cover any subset of size $2^{k-1}$ that does not contain the one revoked user. We want to add to the cover the subset that includes the most users not already covered. We can add subsets of size $2^{k-1}$, but what is the largest the union of the two can be? If the two subsets intersect (and one is not contained within the other), then the intersection must be $2^{k-2}$ as the second subset must be in the form of Formula (5.6). This is the same size union we would get using disjoint subsets. The same applies to smaller subsets.

A similar argument holds for $t_{\max}(n,2)$. The sum of the sizes of the subsets must add up to $2^k - 2$. The process of subtracting the size of the largest subset will continue as before, only it will stop one step earlier at $2^1 - 2 = 0$. Therefore,

any cover of $n-2$ users will require at least $k-1 = \log_2(n)-1$ subsets, and so $t_{\max}(n,2) \leq \log_2(n)-1$. By Lemma 23, $t(\mathcal{N},\mathcal{R}) = \log_2(n)+h-2$, where $h$ is the height of the least common ancestor of two leaves. Since the ancestor of two leaves must be height at least one, $t(\mathcal{N},\mathcal{R}) \geq \log_2(n)-1$, and the same must be true of $t_{\max}(n,2)$. Therefore, $t_{\max}(n,2) = \log_2(n)-1$.

Formula 4 holds for all revocation schemes. We have the condition that there must be indices such that $\gamma(i_j) = \{u_i\}$ for all $u \in \mathcal{N}$. So we can always cover $\mathcal{N} \setminus \mathcal{R}$ with one subset when $|\mathcal{R}| = n-1$: $t_{\max}(n,n-1) = 1$. The same is true when $|\mathcal{R}| = n-2$ in a Complete Forest Revocation Scheme. Just as we can find a scheme where any two revoked users are siblings, we can also find a scheme where any two privileged users are siblings. The index for their parent is sufficient for the cover. Therefore $t_{\max}(n,n-2) = 1$. □

This last argument can be generalised to give a bound on $t_{\max}(n,r)$ for any $r$:

*Lemma* 59. Let $RS$ be a Complete Forest Revocation Scheme on $n = 2^k$ users. Then $RS$ has:

$$t_{\max}(n,r) \leq \left\lfloor \frac{n-r+1}{2} \right\rfloor. \tag{5.7}$$

*Proof.* We have from Corollary 6 that $t_{\max}(n,r) \leq n-r$, which comes from the fact that each user owns one key only known by himself and the centre. In a Complete Forest we have the property that any pair of users are siblings on one of the trees and hence there is an index in the scheme that covers only those two users. The centre can therefore divide all privileged users into pairs (in any order) and use the indices for the subsets of users of size 2 (with one extra if there is an odd number). This gives:

$$t_{\max}(n,r) \leq \left\lfloor \frac{n-r+1}{2} \right\rfloor. \qquad \square$$

Note that $t^{RS}_{\max}(n,n-2) = 1$ and Formula (5.7) would still be true of a Complete Forest Revocation Scheme even if we did not stipulate that it be the union of Complete Subtree Revocation Schemes. It would be equally possible to create a scheme with the same property by way of a union of other schemes. The reason we use the Complete Subtree is because we want to

minimise $t_{\max}(n,r)$ for all values of $r$, and these bounds mainly give small $t(\mathcal{N},\mathcal{R})$ for large values of $r$.

Lemma 59 uses the fact that we can combine the covers of two (or more) pairs of privileged users. Unfortunately, we cannot do the same when we have more than 1 pair of revoked users. We can find the covers when only one pair is revoked at a time, and we have the formula in Lemma 58 that says $t(\mathcal{N},\mathcal{R}) = \log_2(n) - 1$ (when $|\mathcal{R}| = 2$), but there is no simple way to combine the covers. What this means is that although the scheme itself can find a cover, there is no corresponding formula like those in Lemma 59 for $t_{\max}(n,r)$ when $r = 4, 6, 8, \ldots$. There is one more bound we can place on $t(\mathcal{N},\mathcal{R})$. It uses both formulae for $t_{\max}(n,r)$ of unions of schemes we proved at the start of the chapter.

*Lemma* 60. Let $RS_1$ be a Complete Forest Revocation Scheme on $n$ users, and $RS_2$ be a Complete Forest Revocation Scheme on $2n$ users, both constructed as we have outlined. Then $RS_2$ has:

$$t^{RS_2}_{\max}(2n,r) \leq \max_{\substack{0\leq r_1,r_2\leq r\\ r_1+r_2=r}} \Big(t^{RS_1}_{\max}(n,r_1)+t^{RS_1}_{\max}(n,r_2)\Big), \qquad \text{for all } 0 \leq r \leq 2n. \quad (5.8)$$

*Proof.* In order to form a Complete Forest on $2n$ users, we take the disjoint union of the scheme on $n$ users with itself. If we call the resulting scheme $RS_3$, then by Theorem 52 we have:

$$\begin{aligned} t^{RS_3}_{\max}(2n,0) &= 1 \\ t^{RS_3}_{\max}(2n,r) &= \max_{\substack{0\leq r_1,r_2\leq r\\ r_1+r_2=r}} \Big(t^{RS_1}_{\max}(n,r_1) + t^{RS_1}_{\max}(n,r_2)\Big), \qquad \text{for all } 1 \leq r \leq 2n. \end{aligned}$$

We combine this with a union of $n$ Complete Subtree Revocation Schemes (the properties of these schemes do not affect this proof), which we will call $RS_4$. $RS_2$ is just the union of $RS_3$ and $RS_4$ and by Lemma 43, we have:

$$\begin{aligned} t^{RS_2}_{\max}(2n,r) &\leq \min(t^{RS_3}_{\max}(2n,r), t^{RS_4}_{\max}(2n,r)) \\ &\leq \max_{\substack{0\leq r_1,r_2\leq r\\ r_1+r_2=r}} \Big(t^{RS_1}_{\max}(n,r_1) + t^{RS_1}_{\max}(n,r_2)\Big), \qquad \text{for all } 0 \leq r \leq 2n. \quad \square \end{aligned}$$

Formula (5.8) can be used to establish a recursive bound on $t_{\max}(n,r)$. In order to get as close a bound on $t_{\max}(n,r)$ as possible, we will combine this with the bound of Formula (5.7) and the equations in Lemma 58.

Let $B(n,r)$, for any $n = 2^k \geq 2^2$ and $0 \leq r \leq n$, be defined as follows:

$$\begin{aligned} B(n,r) &= min\left\{ \left\lfloor \frac{n-r+1}{2} \right\rfloor, X, Y \right\}, \qquad \text{where} \\ X &= \begin{cases} \text{value from Lemma 58} & \text{if } r \in \{0,1,2,n-2,n-1,n\} \\ \infty & \text{otherwise} \end{cases} \\ Y &= \max_{\substack{0 \leq r_1, r_2 \leq r \\ r_1 + r_2 = r}} \Big( B(n/2, r_1) + B(n/2, r_2) \Big). \end{aligned}$$

We can show that $B(n,r)$ is an upper bound for $t_{\max}(n,r)$ by looking at the three possibilities when evaluating it. If $B(n,r) = \lfloor \frac{n-r+1}{2} \rfloor$, then $B(n,r) \geq t_{\max}(n,r)$ by Lemma 59. If $B(n,r) = X$, then we will have $B(n,r) \geq t_{\max}(n,r)$ by Lemma 58 (actually have equality). If $B(n,r) = Y$, then $B(n,r)$ is an upper bound of $t_{\max}(n,r)$ if $B(n/2,r)$ is an upper bound for $t_{\max}(n/2,r)$. Assuming that $B(n/2,r) \geq t_{\max}(n/2,r)$ then:

$$\begin{aligned} B(n,r) &= \max_{\substack{0 \leq r_1, r_2 \leq r \\ r_1 + r_2 = r}} \Big( B(n/2, r_1) + B(n/2, r_2) \Big) \\ &\geq \max_{\substack{0 \leq r_1, r_2 \leq r \\ r_1 + r_2 = r}} \Big( t^{RS_1}_{\max}(n/2, r_1) + t^{RS_1}_{\max}(n/2, r_2) \Big) \\ &\geq t_{\max}(n,r). \end{aligned}$$

So $B(n,r)$ is an upper bound for $t_{\max}(n,r)$ if $B(n/2,r)$ is an upper bound for $t_{\max}(n/2,r)$. Similarly, $B(n/2,r)$ is an upper bound for $t_{\max}(n/2,r)$ if $B(n/4,r)$ is an upper bound for $t_{\max}(n/4,r)$. Eventually we get to the case $n = 4$, for which it is simple to calculate $t_{\max}(n,r)$ by exhaustive enumeration. As we can see in Table 5.4, $B(4,r) \geq t_{\max}(4,r)$ (we actually have equality). So $B(n,r) \geq t_{\max}(n,r)$ for all powers of two greater than or equal to 4.

The bound also has the added advantage that we can form a cover of any $\mathcal{N} \setminus \mathcal{R}$ in the Complete Forest Revocation Scheme that will be no bigger that $B(n,r)$, where $|\mathcal{N}| = n$ and $|\mathcal{R}| = r$. Both Lemma 58 and Lemma 59 describe how to form a cover which is at least as small as their respective bounds. If this cover is greater than $B(n,r)$, then $\mathcal{N} \setminus \mathcal{R}$ can be partitioned into the two user set of the component schemes. Repeating this process will eventually result in a cover which is less than or equal in size to $B(n,r)$.

It is unsurprising that we get equality when $n = 4$, as all the values are calculated form the formulae in Lemma 58, which all have equality. When

| $r$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $t^{CS}_{\max}(4,r)$ | 1 | 2 | 2 | 1 | 0 |
| $B(4,r)$ | 1 | 2 | 1 | 1 | 0 |
| $t^{RS}_{\max}(4,r)$ | 1 | 2 | 1 | 1 | 0 |

Table 5.4: $t_{\max}(n,r)$ for Complete Forest Revocation Scheme and Complete Subtree Revocation Scheme ($n = 4$)

| $r$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $t^{CS}_{\max}(8,r)$ | 1 | 3 | 4 | 4 | 4 | 3 | 2 | 1 | 0 |
| $B(8,r)$ | 1 | 3 | 2 | 3 | 2 | 2 | 1 | 1 | 0 |
| $t^{RS}_{\max}(8,r)$ | 1 | 3 | 2 | 3 | 2 | 2 | 1 | 1 | 0 |

Table 5.5: $t_{\max}(n,r)$ for Complete Forest Revocation Scheme and Complete Subtree Revocation Scheme ($n = 8$)

$n = 8$, these formula are insufficient to calculate $t^{RS}_{\max}(8,r)$ for all $r$. The values we get for $B(8,r)$ with equations (5.7) and (5.8) still equal $t^{RS}_{\max}(8,r)$ (Table 5.5). However, when we get to $n = 16$ there are some discrepancies. There is a difference of 1 between $B(16,r)$ and $t^{RS}_{\max}(16,r)$ when $r = 4$ and $r = 6$ (Table 5.6). Unfortunately, this method of calculating a bound on $t^{RS}_{\max}(n,r)$ will cause any differences to multiply. Each iteration assumes that for the smaller value of $n$ in Formula (5.8) (i.e. $n/2$) $B(n,r) = t^{RS}_{\max}(n,r)$, rather than just being an upper bound.

We do at least get a reasonable upper bound on $t^{RS}_{\max}(n,r)$, and we can see a clear improvement over the Complete Subtree Revocation Scheme. Just from Equation (5.7) we get roughly half the bandwidth cost (for large $r$). The value of $t^{CS}_{\max}(n,r) = \min(n/2, n-r)$ for $r \geq n/4$, which is almost twice the value of $\lfloor \frac{n-r+1}{2} \rfloor$. But an increase in storage by a factor of $n$ is almost certainly prohibitive.

There are a few reasons why this bound differs from the actual maximum $t^{RS}_{\max}(n,r)$. The recursive relation (5.8) uses the fact that a Complete Forest with $2n$ users contains the disjoint union of two Complete Forest schemes, each with $n$ users. So any set of privileged users can be split into two sets, and

| $r$ | 0 | 1 | 2 | 3 | **4** | 5 | **6** | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t^{CS}_{\max}(16, r)$ | 1 | 4 | 6 | 7 | **8** | 8 | **8** | 8 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| $B(16, r)$ | 1 | 4 | 3 | 5 | **6** | 5 | **5** | 5 | 4 | 4 | 3 | 3 | 2 | 2 | 2 | 1 | 0 |
| $t^{RS}_{\max}(16, r)$ | 1 | 4 | 3 | 5 | **5** | 5 | **4** | 5 | 4 | 4 | 3 | 3 | 2 | 2 | 2 | 1 | 0 |

Table 5.6: $t_{\max}(n, r)$ for Complete Forest Revocation Scheme and Complete Subtree Revocation Scheme ($n = 16$)

the cover found in the corresponding schemes. However, the Complete Forest is the union of these with $n$ other Complete Subtree Revocation Schemes. Any of these may reduce the size of any cover, including those that give the maximum in the smaller schemes. The earlier bound of (5.7) uses the fact that any pair of privileged users are siblings in at least one tree. But it does not take into account that if we have enough pairs of siblings, we might be guaranteed that two of the parents of pairs of siblings might be siblings in one of the trees as well. This would mean that four privileged users share a key exclusively instead of only two, and hence $t(\mathcal{N}, \mathcal{R})$ is reduced. So whenever the bound differs from $t_{\max}(n, r)$, even by just one, this will trickle through the calculations of $B(n, r)$ for all larger values of $n$.

In this section, we have seen an application of the union, and disjoint union, of schemes. The resulting scheme does a very low bandwidth cost. This comes with the cost of prohibitively high storage, making it impractical in most cases. We also have the problem of not being able to analyse the bandwidth. We only know bounds on $t_{\max}(n, r)$, we do not know exactly how much of an improvement the Complete Forest Revocation Schemes is over the Complete Subtree Revocation Schemes. In the next section, we will construct a variant on the Complete Forest Revocation Scheme that avoids these problems.

## 5.3 Partial Forest

We wish to construct a scheme that has lower bandwidth than the Complete Subtree Revocation Scheme, but without the extremely high storage of the Complete Forest. Fortunately, there is a clear middle ground between the

Complete Subtree Revocation Scheme and the Complete Forest of $n-1$ Trees Revocation Scheme. Instead of having the property that any pair of users chosen from the entire set of users are siblings in one tree we are going to make a weaker claim that does not require as many trees. We are going to pick some level in the binary tree between the root and the $2^{nd}$ last level from the bottom (nodes at distance 2 from the leaves). We will consider all the users descended from a particular node at this level to be part of the same group. We let $g$ be the number of users in this group ($g$ is necessarily a power of 2). This partitions the set of users into $g' = n/g$ groups of $g$ ($g'$ is also a power of 2). The requirement of the forest is that any pair within a group are siblings in at least one tree. We say nothing of pairs of leaves chosen from different groups.

*Definition* 61. Let $RS = (\mathcal{N}, \Omega', \gamma')$ be the union of $X$ Complete Subtree Revocation Schemes, $RS_1 = (\mathcal{N}, \Omega, \gamma_{f_1}), \ldots, RS_X = (\mathcal{N}, \Omega, \gamma_{f_X})$, with $|\mathcal{N}| = n$. Let the subsets $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_g$ be a partition of $\mathcal{N}$ into $g'$ equally sized sets (each of size $g$, where $2 < g < n$). We say that $RS$ is a *Partial Forest Revocation Scheme* on $g-$*subsets* if for all pairs of users $u_{i_1}, u_{i_2} \in \mathcal{N}$ such that $u_{i_1}, u_{i_2}$ are in the same partition $\mathcal{N}_j$, there exists an index $i \in \Omega'$ such that

$$\gamma'(i) = \{u_{i_1}, u_{i_2}\}.$$

Let each partition of $g$ users be a group. First of all, since the number of users in each group and the number of groups are both whole numbers whose product is a power of 2 ($n = 2^k$), they must both also be powers of 2. Second, we can show the requirement can be satisfied by a union of $g-1$ schemes using the same argument to show the same was true for the Complete Forest. There are $\binom{g}{2}$ distinct pairs of users from any one group, and each scheme can only have at most $g/2$ pairs of siblings (again from any one group), so $\frac{g(g-1)}{2} / \frac{g}{2} = g - 1$.

So how do we go about constructing a Partial Forest? The construction is a very simple modification of the Complete Forest algorithm. To create a Complete Forest with $2n$ users, we first got a Complete Forest with $n$ users. We took a disjoint union of this with itself. This gives us a scheme with $2n$ users comprising of two halves, where each half has all the properties of a

Complete Forest on $n$ users. More schemes are added to give it the properties of a Complete Forest on $2n$ users, but it is just his first operation that we will use in the Partial Forest.

The construction goes as follows: Construct a Complete Forest on $g$ users (union of $g-1$ Complete Subtree Revocation Schemes). We repeatedly take a disjoint union of this scheme with itself, doubling the number of users each time, until we have $n$ users. This is the same process as the construction for the Complete Forest up until we have $g-1$ trees of length $g$ (first $\log_2(g)$ steps), and it is only the first half of the process for the rest (taking the disjoint union, but not adding new schemes). The output is a union of $g-1$ Complete Subtree Revocation Schemes with $n$ users.

It is simple to show this results in a scheme that satisfies the requirements of a Partial Forest. The Complete Forest on $g$ users has the property we want: any pair of users from the set of size $g$ are siblings in one tree. But the number of users in this scheme is $g$ and we need the same property to hold in a scheme with $n$ users, where $n$ is a power of 2 greater than $g$ (if $g = n$ then the constructed scheme is a Complete Forest Revocation Scheme). In taking the disjoint union with itself, not only do we get double the users, but all properties of the first half, $\{u_1, u_2, \ldots, u_g\}$, are present in the second half, $\{u_{g+1}, u_{g+2}, \ldots, u_{2g}\}$. So after $\log_2(g')$ steps we have a disjoint union of $g'$ Complete Forest Revocation Schemes on $g$ users. The $g'$ distinct sets of $g$ users each have the property required of a Partial Forest because there is a component Complete Forest on each of the $g'$ sets of users in the overall scheme.

In the definition of a Partial Forest, we restricted the values of $g$ to be greater than 2 and less than $n$. As has already been said, we do not include $g = n$ as we wish to distinguish a Partial Forest and a Complete Forest Revocation Scheme. If $g = 1$ then the requirement will be satisfied with any revocation scheme. There are no pairs of users that can be chosen from a group of size one, so this puts no restriction on the scheme. The case of $g = 2$ is also trivial, as the property is satisfied by a single tree if we define the groups to be those pairs of users who are siblings. There is only one option when choosing pairs of users from a group of size 2, and by definition those users are siblings. This

is why we choose the defining node of a group to be at least at a distance 2 from the leaves. We will look at the first interesting case, $g = 4$.

The construction of the Complete Forest of Trees when $n = 4$ results in a scheme with the following leaf lists: $[u_1, u_2, u_3, u_4], [u_1, u_3, u_2, u_4], [u_1, u_4, u_2, u_3]$. For any $n$ greater than 4 we will need to stretch these schemes. A group will be a subtree rooted 2 levels above the leaves (so it must have $2^2 = 4$ users). To get the property that any pair of users from the same group will be siblings in at least one of the trees, we only need the following trees:

$$\begin{aligned}
\text{Tree 1} &= [u_1, u_2, u_3, u_4, \quad u_5, u_6, u_7, u_8, \quad \ldots, \quad u_{n-3}, u_{n-2}, u_{n-1}, u_n], \\
\text{Tree 2} &= [u_1, u_3, u_2, u_4, \quad u_5, u_7, u_6, u_8, \quad \ldots, \quad u_{n-3}, u_{n-1}, u_{n-2}, u_n], \\
\text{Tree 3} &= [u_1, u_4, u_2, u_3, \quad u_5, u_8, u_6, u_7, \quad \ldots, \quad u_{n-3}, u_n, u_{n-2}, u_{n-1}].
\end{aligned}$$

Just looking at the first four columns of leaves, we can see that all $\binom{4}{2}$ pairs from $\{u_1, u_2, u_3, u_4\}$ occur as siblings in one of the trees. This means that what would be the worst case distribution for $n/2$ users with one Complete Subtree (every second user revoked) would only require $t(\mathcal{N}, \mathcal{R}) = n/4$ or half the number of subsets. Of course, this is not necessarily the value of $t_{\max}(n, r)$ for these 3 Trees, as the choice of $\mathcal{R}$ that gives $t_{\max}(n, r)$ in one scheme may not do the same in another. Also, we still have all other values of $r$ to consider.

The formula for $t_{\max}(n, r)$ is relatively straightforward, by the nature of the construction and Theorem 52.

*Theorem* 62. Let $PFRS = (\mathcal{N}, \Omega, \gamma)$ be a Partial Forest Revocation Scheme on $g - subsets$ sets, where $g.g' = n$ and $|\mathcal{N}| = n$. Then:

$$t_{\max}^{PFRS}(n, 0) = 1 \tag{5.9}$$

$$t_{\max}^{PFRS}(2g, r) = \max_{\substack{0 \le r_1, r_2 \le \min(r, g) \\ r_1 + r_2 = r}} \left( t_{\max}^{CFRS}(g, r_1) + t_{\max}^{CFRS}(g, r_2) \right) \tag{5.10}$$

$$\text{for all } 1 \le r \le 2g$$

$$t_{\max}^{PFRS}(2n, r) = \max_{\substack{0 \le r_1, r_2 \le \min(r, n) \\ r_1 + r_2 = r}} \left( t_{\max}^{PFRS}(n, r_1) + t_{\max}^{PFRS}(n, r_2) \right), \tag{5.11}$$

$$\text{for all } 1 \le r \le 2n \text{ and } n > g.$$

where $CFRS$ is the Complete Forest Revocation Scheme on $g$ users.

*Proof.* A Partial Forest Revocation Scheme on 2 sets, each of size $g$ is comprised of the disjoint union of a Complete Forest Revocation Scheme on $g$ users

($CFRS$), with itself. By Theorem 52, we have Formulae (5.9) and (5.10). If $n > g$, then we take the disjoint union of the above scheme with itself, several times, until we have $2n$ users. This means that for any $n > g$, a Partial Forest with $2n$ users is the disjoint union of a Partial Forest with $n$ users, with itself. Therefore we can use Theorem 52 to get Formula (5.11). □

This gives us a way to calculate $t_{\max}(n, r)$ for a Partial Forest on $g - subsets$, but only if we know $t_{\max}(g, r')$ with a Complete Forest for all $0 \leq r' \leq g$. But in the previous section we could only find $t_{\max}(n, r)$ by exhaustive search, the formulae we found were only upper bounds. That means we can only calculate $t_{\max}(n, r)$ for a Partial Forest on $4, 8$ or $16 - subsets$. If we substituted the upper bound for $t_{\max}(n, r)$ with a Complete Forest, $B(n, r)$, into Formula (5.10), then we would get an upper bound for $t_{\max}(n, r)$ with a Partial Forest. This would give us some idea of the performance of the scheme for larger $g$.

We can get another bound on $t_{\max}(n, r)$ in the Partial Forest, which stems directly from the definition. The result is an equation similar to Formula (5.7) for the Complete Forest.

*Lemma* 63. Let $PFRS = (\mathcal{N}, \Omega, \gamma)$ be a Partial Forest Revocation Scheme on $g - subsets$ sets, where $g.g' = n$, $|\mathcal{N}| = n$ and $g' \geq 2$. Then:

$$t_{\max}^{PFRS}(n, r) \leq \left\lfloor \frac{n - r}{2} \right\rfloor + \frac{g'}{2}. \tag{5.12}$$

*Proof.* Let $\mathcal{R}$ be any subset of $\mathcal{N}$ of size $r$ that gives $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. By the definition of a Partial Forest, we can cover $\mathcal{N} \setminus \mathcal{R}$ as follows: Partition $\mathcal{N} \setminus \mathcal{R}$ into the various groups in $PFRS$. In each group, use 1 index per pair of privileged users (which we can do as it is a Partial Forest), with one extra if there is a privileged user left over. If $t_2$ is the number of users left over, then there are $t_1 = \frac{n-r-t_2}{2}$ pairs of users in all, which means the size of the cover can be written as:

$$t_1 + t_2 = \frac{n - r - t_2}{2} + t_2 = \frac{n - r}{2} + \frac{t_2}{2}.$$

Since $n$ and $r$ are fixed, the size of the cover is dependent on $t_2$. Because there are only $g'$ groups, there can be at most $g'$ users left over. We can go further

and say that there can be at most $g'$ users left over when $n-r$ is even and at most $g'-1$ users left over when $n-r$ is odd. This is because $g'$ is even (a power of 2). So summing $g'$ odd numbers can only give an even number. When $n-r$ is odd, at least one of the groups must have an even number of privileged users. Therefore:

$$t_2 \leq g' - (n-r \mod 2).$$

So an upper bound for the size of the cover is:

$$\begin{aligned} t_1 + t_2 &= \frac{n-r}{2} + \frac{t_2}{2} \\ &\leq \frac{n-r}{2} + \frac{g' - (n-r \mod 2)}{2} \\ &= \frac{n-r-(n-r \mod 2)}{2} + \frac{g'}{2} \\ &= \left\lfloor \frac{n-r}{2} \right\rfloor + \frac{g'}{2}. \end{aligned}$$

The value of $t(\mathcal{N}, \mathcal{R})$ is less than or equal to $t_1 + t_2$ as this is not necessarily the way to find the minimal cover of $\mathcal{N} \setminus \mathcal{R}$. So:

$$t(\mathcal{N}, \mathcal{R}) \leq \left\lfloor \frac{n-r}{2} \right\rfloor + \frac{g'}{2}.$$

Since $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$, we have:

$$t_{\max}(n, r) \leq \left\lfloor \frac{n-r}{2} \right\rfloor + \frac{g'}{2}. \qquad \square$$

We can use this to show that the Partial Forest Method has $t_{\max}(n, r)$ smaller than that of the Complete Subtree Revocation Scheme. For $r$ in the range $n/2 \leq r \leq n$ we have that $t_{\max}^{CSRS}(n, r) = n - r$. In the same range, $t_{\max}^{PFRS}(n, r)$ is bounded above by Formula (5.12), or just:

$$t_{\max}^{PFRS}(n, r) \leq \frac{n-r}{2} + \frac{g'}{2}.$$

If we consider these two as functions of $r$ (which is how we plot $t_{\max}(n, r)$), then they intersect at

$$\begin{aligned} \frac{n-r}{2} + \frac{g'}{2} &= n - r \\ \text{i.e.} \quad r &= n - g'. \end{aligned}$$

The line that serves as a bound for $t_{\max}(n,r)$ for the Partial Forest has a greater slope $(-1/2)$ than that of the Complete Subtree $(-1)$. Therefore

$$t_{\max}^{PFRS}(n,r) < t_{\max}^{CSRS}(n,r),$$

for $n/2 \leq r < n - g'$. This is a non-empty range, as the biggest $g'$ can be is $n/4$ when we have a Partial Forest on $4 - subsets$.

So the Partial Forest Revocation Scheme has $t_{\max}(n,r)$ smaller than that of the Complete Subtree Revocation Scheme, but the reduction is not as good as that with the Complete Forest Revocation Scheme. The advantage of the Partial Forest lies in the storage requirement. In calculating the storage for the Complete Forest, we counted distinct subsets in the trees added at each step of the construction. We have already shown that the Partial Forest is just a slight modification of this construction. Namely, we create a Complete Forest with $g$ users (union of $g-1$ Complete Subtree Revocation Schemes), and add no further schemes. So we can use the same derivation, producing a sum that stops at $g-1$ rather than extending to $n-1$.

*Corollary* 64. Let $PFRS = (\mathcal{N}, \Omega, \gamma)$ be a Partial Forest Revocation Scheme on $g - subsets$ sets, where $g = 2^l$, $n = 2^k$ and $|\mathcal{N}| = n$. Then:

$$|U|_{\max} = (k+1) + g(l-2) + 2. \tag{5.13}$$

*Proof.* In Lemma 57 we saw that each scheme added in step $i$ of the construction of a Complete Forest increases the storage of every user by $i$. There are $2^i$ such schemes added in step $i$. The construction for a Partial Forest mimics that of a Complete Forest, until there are $g-1$ schemes and then no more are added. This means that there are $l-1$ steps in the construction, instead of $k-1$ ($\sum_{i=0}^{l-1} 2^i = g-1$). So we have a total storage requirement of:

$$\begin{aligned} |U|_{\max} &= k+1+\sum_{i=1}^{l-1} i2^i \\ &= (k+1)+g(l-2)+2. \end{aligned}$$

□

So we have gone from a storage of $\mathcal{O}(n\log(n))$ to one of $\mathcal{O}(\log(n)+g\log(g))$. If $g$ is fixed, then the storage has a fixed extra cost on top of that of the Complete Subtree, no matter how large $n$ is. For $g=4$, $g=8$ and $g=16$ the

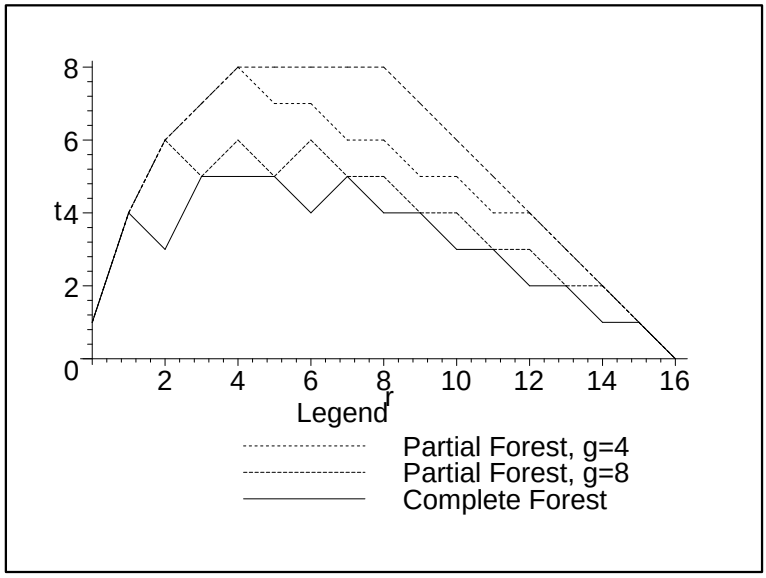


Figure 5.2: $t_{\max}(n, r)$ for Complete and Partial Forests Revocation Schemes, $n = 16$. Top to bottom: $g = 2$ (Complete Subtree), $g = 4$, $g = 8$ and Complete Forest, $t_{\max}(16, r)$ (found by exhaustive search over all $\mathcal{R}$).

extra cost is 2, 10 and 34 respectively. What this means is that for $n$ users, the storage for the Complete Subtree Revocation Scheme is $\log_2(n) + 1$, the storage for a Partial Forest on $4 - subsets$ is $\log_2(n) + 1 + 2$, on $8 - subsets$ is $\log_2(n) + 1 + 10$, on $16 - subsets$ is $\log_2(n) + 1 + 34$, etc.

Some graphs of $t_{\max}(n, r)$ are plotted in Figures 5.2 and 5.3. Figure 5.2 plots $t_{\max}(16, r)$ of all possible Partial Forests (from the Complete Subtree to the Complete Forest) for $n = 16$. All values are calculated by exhaustive search. We can clearly see the step function that results from Formula (5.12), but the Partial Forests also performs better than the Complete Subtree outside the range given above. But the figures are too small to make much comparisons between the schemes. In Figure 5.3 we have a much larger population, $n = 512$. Formulae (5.9), (5.10) and (5.11) were used to find $t_{\max}(512, r)$ for the Partial Forests, as well as Formula (4.1) for the Complete Subtree. As $t_{\max}(512, r)$ cannot be calculated for the Complete Forest Revocation Scheme, instead we have plotted the upper bound $B(n, r)$.

What we see is the step function (Formula (5.12)), get closer to $\frac{n-r}{2}$ as $g$ increases. The graphs get close to $B(512, r)$ quite quickly. There is very little difference between $t_{\max}(512, r)$ for the Partial Forest with $g = 16$, and

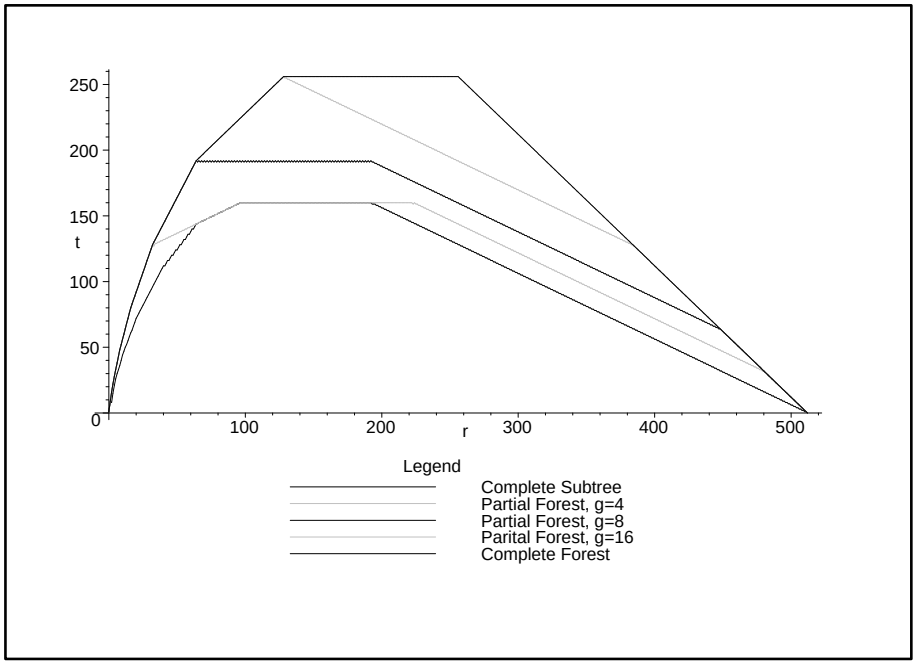


Figure 5.3: Complete and Partial Forests Revocation Schemes, $n = 512$. Top to bottom: $t_{\max}(512, r)$ for $g = 2$ (Complete Subtree), $g = 4$, $g = 8$, $g = 16$, and $B(512, r)$ for Complete Forest.

$B(512, r)$. This comparison may not seem fair as we are comparing a upper bound with a known formula. But at the very least, for large values of $r$, $t_{\max}(n, r)$ is not likely to stray too far from $B(n, r) = \lfloor \frac{n-r+1}{2} \rfloor$. We also see a very distinctive levelling off of the graphs when they reach the highest value of $t_{\max}(512, r)$. With the exception of $g = 4$, all graphs remain close to a constant height for a range of $r$, when they reach the maximum. In Chapter 4 we showed that $t_{\max}(n, r) = n/2$ for the Complete Subtree Revocation Scheme with $n/4 \leq r \leq n/2$ (follows from Corollary 26). To describe the maximum value of $t_{\max}(n, r)$ in any of the Partial Forests, we need to look at the maximum value in the underlying Complete Forest.

The values for $t_{\max}(n, r)$ in a Complete Forest on 4 users are given in Table 5.4. The highest value of $t_{\max}(4, r)$ is 2 when $r = 1$. So when we have the disjoint union of $n/4$ such schemes, we will get the maximum value of $t_{\max}(n, r)$ when $r = 1 \times (n/4)$. Since $r = 1$ was the only value where $t_{\max}(4, r) = 2$, there is only one value of $r$ where we get $t_{\max}(n, r) = n/4$ in a Partial Forest on $4 - subsets$. In a Complete Forest on 8 users, we have

two values of $r$ that give the maximum value of $t_{\max}(n, r)$. If $r = 1$ or 3, $t_{\max}(8, r) = 3$. So in the Partial Forest that is comprised of the disjoint union of $n/8$ Complete Forests, we get $t_{\max}(n, r) = 3n/8$ if $r = 1 \times (n/8)$ or $3 \times (n/8)$. These are the extreme values of $r$ where we get the maximum, but they are not the only ones. Any $r$ that is the sum of $n/8$ integers from $[1, 3]$ will give the same value of $t_{\max}(n, r)$. Starting at $r = n/8$, the next highest value can only be $r = n/8 + 2$, gotten by replacing one of the 1's in the sum by a 3. As this is all we can do to $r$ and still have $t_{\max}(n, r) = 3n/8$, only even values of $r$ in the range $n/8 \leq r \leq 3n/8$ can satisfy this equation. Hence we get a sawtooth function:

$$t_{\max}(n, r) = \begin{cases} 3n/8 & \text{if } n/8 \leq r \leq 3n/8 \text{ and } r \text{ is even} \\ 3n/8 - 1 & \text{if } n/8 \leq r \leq 3n/8 \text{ and } r \text{ is odd.} \end{cases}$$

In a Complete Forest on 16 users, the maximum value is $t_{\max}(16, r) = 5$ for any of $r = 3, 4, 5, 7$. So the extreme values of $r$ that give maximum $t_{\max}(n, r)$ are $t_{\max}(n, 3n/16) = 5n/16$ and $t_{\max}(n, 7n/16) = 5n/16$. Any $r$ inside these two values will also give the same value provided it can be written as the sum of $n/16$ integers from $[3, 4, 5, 7]$. But for how many values of $r$ does his hold? We can re-phrase the problem by subtracting 3 from each of the integers in the list, which means we have to subtract $3 \times (n/16)$ from the range. So how many values of $r$ in the range $\{0, \ldots, 4n/16\}$ can be written as the sum of $n/16$ integers from $[0, 1, 2, 4]$? There is at least one value that cannot, $4n/16-1$. We can get to within $\pm 1$ with $4 \times (n/16-1)+2 = 4n/16-2$ and $4 \times (n/16) = 4n/16$, but any other choice of integers will give a sum $< 4n/16 - 2$. It turns out that this is the only value in the range that is not the sum of $n/16$ integers from $[0, 1, 2, 4]$.

*Lemma* 65. Let $g'$ be a positive integer. Then $\forall r \in \{0, 1, \ldots, 4g' - 2, 4g'\}$, $\exists \mathbf{v} \in [0, 1, 2, 4]^{g'}$ such that $\sum_{i=1}^{g'} \mathbf{v}_i = r$.

*Proof.* Proof by induction on $g'$. Case $g' = 1$ amounts to showing that for any $r \in \{0, 1, 2, 4\}$ is equal to the sum of one integer from $[0, 1, 2, 4]$. This is obvious. Assume hypothesis is true for $g' = k$. Need to show true for $g' = k+1$. To do this, we must show that for any $r \in \{0, 1, \ldots, 4k + 2, 4k + 4\}$, $\exists \mathbf{v} \in [0, 1, 2, 4]^{k+1}$ such that $\sum_{i=1}^{k+1} \mathbf{v}_i = r$. But by the induction hypothesis we sum

up to any $r \in \{0, 1, \ldots, 4k-2, 4k\}$ with $k$ integers from $[0, 1, 2, 4]$. By adding an extra element, 0, onto each of these vectors, we get the same sums but with $g' = k+1$ elements. The remaining cases ($r \in \{4k-1, 4k+1, 4k+2, 4k+4\}$) can be treated as special cases:

$$\begin{aligned}
\underbrace{4+\ldots+4}_{k-1}+1+2 &= 4k-1 \\
\underbrace{4+4+\ldots+4}_{k}+1 &= 4k+1 \\
\underbrace{4+4+\ldots+4}_{k}+2 &= 4k+2 \\
\underbrace{4+4+\ldots+4}_{k}+4 &= 4k+4. \qquad \square
\end{aligned}$$

Any $r \in \{0, \ldots, 4n/16-2, 4n/16\}$ can be written as the sum of $n/16$ integers from $[0, 1, 2, 4]$, which means any $r \in \{3n/16, \ldots, 7n/16-2, 7n/16\}$ can be written as the sum of $n/16$ integers from $[3, 4, 5, 7]$. So $t_{\max}(n, r)$ for the Partial Forest on $16-$*subsets* is at its highest for these values.

These points and Formula (5.12) explains how the Partial Forests schemes have smaller $t_{\max}(n, r)$ than the Complete Subtree Revocation Scheme. But as we increase the size of the Partial Forest (increase $g$, and hence the number of Complete Subtrees schemes, $g-1$), there is less and less of an improvement. For example, the height of the step function Formula (5.12) over that of the Complete Forest is dependent only on $g'$ which is halved each time we double $g$. While the improvements in $t_{\max}(n, r)$ are getting less and less for larger Partial Forests, the costs in storage grow exponentially. Since $|U|_{\max} = (k+1) + g(l-2) + 2$, the storage grows with $2^l l$ where $l = 1, 2, \ldots$ is the height of the nodes defining the groups. We can see this growth in Table 5.7. We will discuss how to go about finding a good middle ground for this trade-off in Chapter 7 when we compare all schemes.

The one remaining factor to consider when comparing these schemes is the complexity of finding a cover. The Complete Subtree Revocation Scheme has a very straightforward algorithm, but the algorithm given at the start of the chapter (for any union of schemes) is more complex. What may cause problems is the fact that in order to cover any $\mathcal{N} \setminus \mathcal{R}$, the centre needs to compile the set of all indices $i \in \Omega$ such that $\gamma(i) \in \mathcal{N} \setminus \mathcal{R}$. For large $\mathcal{R}$,

| $l$ | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $g$ | | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
| $\|U\|_{\max}$ | $n = 16$ | 5 | 7 | 15 | 39 | n/a | n/a | n/a | n/a | n/a |
| $\|U\|_{\max}$ | $n = 512$ | 10 | 12 | 20 | 44 | 108 | 268 | 652 | 1548 | 3596 |

Table 5.7: Storage for Partial Forests on $g = 2^l - subsets$

this set will be manageable, but if $\mathcal{R}$ is small then the set of privileged users is large. The set of indices we need for just one of the schemes will be $\mathcal{O}(n)$ (it will be most of the nodes in a tree that has $2n - 1$). Granted that most subsets will be duplicated in the various schemes, but the combined set will still be quite large. It will be searching through this list (which is done several times in the algorithm) that will require the bulk of the time taken to find the cover. So for the Partial Forests Methods with high $g$ and the Complete Forest Revocation Scheme, we require a centre with either the ability to operate on very large sets, or that can tolerate a long delay before broadcasting if the scheme is to be used with small numbers of revoked users.

# Chapter 6

# Subset Difference Revocation Scheme

The main drawback of the Complete Subtree Revocation Scheme is the high bandwidth, with the broadcast key being encrypted as many as $n/2$ times. From $r = n/4$ to $r = n/2$ we have $t_{\max}(n, r) = n/2$ (by Formula (4.1)). The second technique of Naor et al. [23] looks to reduce this at the cost of increasing the storage required. The Subset Difference Revocation Scheme ($SDRS$) increases the number of subsets available, compared to the Complete Subtree Revocation Scheme, in order to reduce the number needed for any cover.

To show exactly how much of an improvement the Subset Difference Revocation Scheme is, we will derive the formula for $t_{\max}(n, r)$, as we did for the Complete Subtree Revocation Scheme. To do this, we will first specify the range of $r$ for which the bound of Naor et al., $t_{\max}(n, r) \leq 2r - 1$, achieves equality. This gives rise to a type of Steiner Tree, a Special Subtree, that is pivotal in finding $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ for all $r$. In the second section, we will define a type of subset $\mathcal{R} \subset \mathcal{N}$, an $M$-type subset, which has the property that $ST(\mathcal{R})$ consists of Special Subtrees. This definition aids in counting the size of covers. The problem of maximising $t(\mathcal{N}, \mathcal{R})$ over all $M$-type subsets can be re-stated as finding $j$-tuples of integers with certain properties. It is by finding these tuples that we derive $t_{\max}(n, r)$. We will also modify the recursive formula of $t_{aver}(n, r)$ for disjoint unions (Formula (5.5)) to apply to the

Subset Difference Revocation Scheme.

Here is a quick review of the Subset Difference Revocation Scheme. $\mathcal{N}$ is the set of users ($|\mathcal{N}| = n$), $\Omega$ is the index set (each index is a placeholder for an establishment key), $T$ is a complete binary tree with $n = 2^k$ leaves and the functions $\gamma$ and $\delta$ determine which users get which keys. From the user's point of view, we have:

- User $u$ is assigned a leaf, $v_l$, on a complete binary tree $T$ ($f(v_l) = u$).
- $\delta(u)$ corresponds to the keys $u$ is given
- $\delta(u) = \{(i,j) : v_i \text{ ancestor of } v_l \text{ and } v_j,\ v_j \text{ not ancestor of } v_l\}$

And for a key $L_{i,j}$:

- Index in $\Omega$ for this key is $(i,j)$
- This index pair corresponds to nodes $v_i$ and $v_j$ on tree $T$
- $\gamma(i,j)$ is the set of users who are given key $L_{i,j}$
- $\gamma(i,j) = \{u : u\text{'s leaf is descended from } v_i \text{ but not from } v_j\}$

When we say that keys are given to the user, this can be either explicitly, or implicitly as described in Chapter 3. The main focus of this chapter will be the bandwidth costs rather than the storage costs.

## 6.1 Maximum Bandwidth

We have already seen from Corollary 17 that in the $SDRS$, $t_{\max}(n,r) \leq 2r-1$, for all $1 \leq r \leq n$. For the case $r = 0$, we have that $t_{\max}(n,0) = 1$, since we added an extra index for this specific case, $\gamma(0,0) = \mathcal{N}$. Also, all subsets that you would find in the Complete Subtree Revocation Scheme are present in the Subset Difference Revocation Scheme (Lemma 16), which means the formula for $t_{\max}(n,r)$ with that scheme (Formula (4.1)) serves as an upper bound for this one. Thus:

$$t_{\max}^{SDRS}(n,r) \leq t_{\max}^{CSRS}(n,r) = r(k-j) - 2(r-2^j),$$

where $n = 2^k,\ j = \lfloor \log_2(r) \rfloor$.

What about the recurrence relations that we found for the disjoint union of schemes in Chapter 5 (Formulae (5.2) and (5.5))? The Subset Difference Revocation Scheme is defined on a binary tree, so it would seem likely that the scheme on $2n$ users consists of the disjoint union of the scheme on $n$ users with itself (as was the case for the Complete Subtree Revocation Scheme). Unfortunately, this is not the case. Let $RS = (\mathcal{N}_3, \Omega_3, \gamma_3)$ be the disjoint union of a Subset Difference Revocation Scheme on two different user sets (of the same size), $(\mathcal{N}_1, \Omega_1, \gamma_1)$ and $(\mathcal{N}_2, \Omega_2, \gamma_2)$. Let $T_1$ and $T_2$ be the two trees that the schemes are defined on. Then the subsets in the disjoint union, $\{\gamma(i,j) : (i,j) \in \Omega_3\}$, are:

- $\gamma_1(i,j) = desc(v_i) \setminus desc(v_j)$, where $v_j$ is a descendant of $v_i$ on tree $T_1$
- $\gamma_2(i,j) = desc(v_i) \setminus desc(v_j)$, where $v_j$ is a descendant of $v_i$ on tree $T_2$
- $\gamma_1(0,0) = \mathcal{N}_1$, from first scheme
- $\gamma_2(0,0) = \mathcal{N}_2$, from second scheme
- $\gamma_3(3,0) = \mathcal{N}_3$, from disjoint union

The above subsets do occur in the Subset Difference Revocation Scheme on $2n$ users, but they do not form the complete list. The only subset mentioned above with users from both $\mathcal{N}_1$ and $\mathcal{N}_2$ is $\mathcal{N}_3$. However, the Subset Difference Revocation Scheme defined for $\mathcal{N}_3$ contains several subsets with users from both $\mathcal{N}_1$ and $\mathcal{N}_2$. Consider the case where $v_i$ is the root (of the tree with $2n$ leaves), and $v_j$ is any node/leaf that is not $v_i$'s child (must have $2n \geq 4$). Clearly $desc(v_i) = \mathcal{N}_3$, as all leaves are descended from the root. The size of $desc(v_j)$ can be at most a quarter of the total number of users (number of descendants of the root is 100%, descendants of a child of the root is 50%, descendants of a grandchild of the root is 25%, ...). As $desc(v_i) \setminus desc(v_j)$ must have more than half the users, there will be users in the subset from both halves $\mathcal{N}_1$ and $\mathcal{N}_2$. But $desc(v_i) \setminus desc(v_j) \neq \mathcal{N}_3$ as there will be at least one user in $desc(v_j)$.

So the Subset Difference Revocation Scheme ($SDRS$) on $2n$ users cannot be formed by taking the disjoint union of $SDRS$ on $n$ users with itself. But

the fact that the two size $n$ schemes are contained in the size $2n$ schemes means that Formulae (5.2) and (5.5) serve as upper bounds (on $t_{\max}(n,r)$ and $t_{aver}(n,r)$ respectively). While it may be possible to modify the formula to work in this case, we will derive an explicit formula for $t_{\max}(n,r)$, just as we did for the Complete Subtree Revocation Scheme. As before, we will look at what choice of $\mathcal{R}$ gives $t(\mathcal{N},\mathcal{R}) = t_{\max}(n,r)$.

But before we do that, we need to describe how to calculate $t(\mathcal{N},\mathcal{R})$ for any $\mathcal{R}$. The size of the cover of $\mathcal{N} \setminus \mathcal{R}$ turns out to be related to how many nodes hang off $ST(\mathcal{R})$, but not as directly as it was for the Complete Subtree Revocation Scheme. If we look at the appearance of one of the subsets output from $\gamma$ in the Subset Difference Revocation Scheme (Figure 6.1), we can see that it can simply be all the leaves descended from one node (subset $\gamma(l,j)$). But the other subset in that diagram, $\gamma(k,i)$, is the set of all leaves descended from two nodes (the right child of $v_k$ and the left-most grandchild of $v_k$). This gives us another way to describe the subset $\gamma(i,j)$: If we consider the path from node $v_i$ to $v_j$, then $\gamma(i,j)$ is the set of all leaves descended from all nodes that hang off this path, including the node that hangs off $v_i$ and excluding any node that hangs off $v_j$. Certainly each such leaf belongs to $\gamma(i,j)$, since it is descended from $v_i$ and not descended from $v_j$. We see here the improvement of the Subset Difference Revocation Scheme over the Complete Subtree Revocation Scheme. In the Complete Subtree Revocation Scheme, each node hanging off $ST(\mathcal{R})$ requires an index in the cover. In the Subset Difference Revocation Scheme several nodes hanging off $ST(\mathcal{R})$ will only require one index in the cover provided they all hang off the same path. We formally define this type of path as follows:

*Definition* 66. Let $\mathcal{R}$ be some subset of leaves of a complete binary tree $T$ (so $ST(\mathcal{R})$ is a subtree of $T$). A path from $v_i$ to $v_j$ is a *Special Path* if the following hold:

- $v_i$ and $v_j$ are in $ST(\mathcal{R})$, with $v_i \neq v_j$
- $v_j$ is a descendant of $v_i$
- All nodes from $v_i$ to $par(v_j)$ (which can be the same node) have one node not in $ST(\mathcal{R})$ hanging off

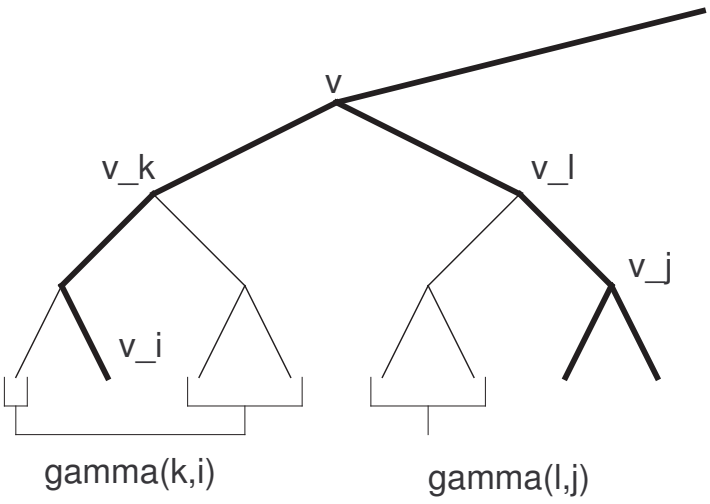


Figure 6.1: Examples of subsets in the Subset Difference Revocation Scheme.

- Either $v_i$ is the root or $par(v_i)$ does not have a node hanging off
- $v_j$ is either a leaf or a node of degree 3 in $ST(\mathcal{R})$.

The set of leaves descended from the nodes hanging off the nodes from $v_i$ to $par(v_j)$ is $S_{i,j}$.

The node $v_j$ is a *null Special Path* if one of the following hold: $v_j$ is the root and has both children in $ST(\mathcal{R})$, $v_j$ is a leaf and $par(v_j)$ has both children in $ST(\mathcal{R})$, or $v_j$ is an internal node and both $v_j$ and $par(v_j)$ have both children in $ST(\mathcal{R})$.

In a Special Path from $v_i$ to $v_j$, $v_i, \ldots, v_{j'} = par(v_j)$ is a maximal path in $ST(\mathcal{R})$ with each node joined to a node not in $ST(\mathcal{R})$, i.e. we cannot extend $v_i, \ldots, v_{j'}$ to a longer path with the same property. As a result, we can make the following statement:

*Lemma* 67. Let $\mathcal{R}$ be a subset of leaves of a complete binary tree. Then the nodes of $ST(\mathcal{R})$ are partitioned into Special Paths and null Special Paths. Furthermore, any node which is not a null Special Path belongs to a unique Special Path.

*Proof.* A null Special Path is either a node of degree 3 (or a leaf) whose parent does not have a node hanging off or else is the root with degree 2. One of the following must be true of any other node, $v$, in $ST(\mathcal{R})$:

1. $v$ has a node not in $ST(\mathcal{R})$ hanging off (either because it is an internal node with degree 2 or the root with degree 1)

2. $v$ is a degree 3 node (or a leaf) whose parent does have a node hanging off

Suppose $v$ is in the form of the first case. Each node in $ST(\mathcal{R})$ has at most one parent in $ST(\mathcal{R})$ (each node in a tree has at most one parent and all paths to the root are in $ST(\mathcal{R})$). Since $v$ is a node in a binary tree and has a node not in $ST(\mathcal{R})$ hanging off, it has exactly one child in $ST(\mathcal{R})$. If this child of $v$ or the parent of $v$ also has a node hanging off, then the same will apply to them. Therefore, there is some unique maximal path of nodes each with one child not in $ST(\mathcal{R})$ that contains $v$. The parent of the node closest to the root on this path is either the root or has a sibling in $ST(\mathcal{R})$. The node furthest from the root on this path has a child with no vertex hanging off. Therefore $v$ belongs to a unique Special Path.

Otherwise, $v$ is a degree 3 node (or a leaf) whose parent does have a node hanging off. By the same argument, $par(v)$ belongs to a unique maximal path of nodes each with one child not in $ST(\mathcal{R})$. Since $v$ can only have one parent, $v$ belongs to a unique Special Path. □

Special Paths are terminated by a degree 3 node or a leaf. In a binary tree, the only other possibility is a node of degree 2, in which case it has another node hanging off (which means the path does not terminate at that node). Although it does not have any node hanging off, we need to define the degree 3 node (or leaf) $v_j$ to be part of the Special Path. Each other node on the path has a node hanging off. If we were to define a Special Path to consist of only those nodes with one child not in $ST(\mathcal{R})$, then we would not know which node hangs off $par(v_j)$ and which is in $ST(\mathcal{R})$. So this node is in the Special Path. As we defined in Section 2.3, the length of a path is the number of edges in the path. As the degree 3 node is included, the length is therefore equal to the number of nodes that hang off. We use $S_{i,j}$ to denote the set of leaves descended from the nodes that hang off the Special Path from $v_i$ to $v_j$ (including $v_i$ but not $v_j$). One of the first things we will prove is that this set is equal to $\gamma(i, j)$.

Before discussing Special Paths any further, we need to prove the relation between the number of Special Paths in $ST(\mathcal{R})$ and the size of the minimal cover. There are a number of results we have to prove first, before we prove this relation. As well as the above mentioned equality between the set of leaves descended from a Special Path from $v_i$ to $v_j$ and $\gamma(i,j)$, we must then prove that there are no nodes common to two distinct Special Paths. We also prove the same for the subsets in a minimal cover, i.e. any cover in a Subset Difference Revocation Scheme is disjoint. The consequence of these results is that $t(\mathcal{N},\mathcal{R})$ is equal to the number of Special Paths in $ST(\mathcal{R})$.

*Lemma* 68. Let $SDRS = (\mathcal{N},\Omega,\gamma)$ be a Subset Difference Revocation Scheme defined on tree $T$. Let $v_i$ to $v_j$ be a Special Path. Then

$$\{f(v_l) : v_l \in S_{i,j}\} = \gamma(i,j)$$

*Proof.* Let $v_l$ be a leaf such that $f(v_l) \in \gamma(i,j)$. Then $v_l$ is a descendant of $v_i$, but not of $v_j$. Since both $v_l$ and $v_j$ are descendants of $v_i$, the least common ancestor of these two nodes is either $v_i$ or some descendant of $v_i$. This node cannot be $v_j$ (or any node that is a descendant of $v_j$) as $v_l$ is not descended from $v_j$. So, it must be on the path from $v_j$'s parent to $v_i$. The least common ancestor has one child that is an ancestor of $v_l$ and one child that is on the path from $v_i$ to $v_j$. Therefore, $v_l$ is a descendant of a node that hangs off the Special Path from $v_i$ to $v_j$, i.e. $v_l \in S_{i,j}$. Conversely, if $v_l \in S_{i,j}$, then it is a descendant of a node that hangs off the Special Path from $v_i$ to $v_j$. The leaf is clearly a descendant of $v_i$. But it cannot be a descendant of $v_j$ as it has an ancestor which is a sibling of an ancestor of $v_j$. Therefore $f(v_l) \in \gamma(i,j)$. So $\gamma(i,j) = \{f(v_l) : v_l \in S_{i,j}\}$. □

The fact that the sets of leaves descended from Special Paths are disjoint follows naturally from the definition.

*Lemma* 69. Let $RS = (\mathcal{N},\Omega,\gamma)$ be a Subset Difference Revocation Scheme defined on tree $T$. Let $v_{i_1}$ to $v_{j_1}$ and $v_{i_2}$ and $v_{j_2}$ be two distinct Special Paths. Then $S_{i_1,j_1} \cap S_{i_2,j_2} = \emptyset$.

*Proof.* $ST(\mathcal{R})$ is a collection of paths from leaves to the root. If any node is in $ST(\mathcal{R})$, then all nodes on the path to the root are also in $ST(\mathcal{R})$. Consequently, if a node, $v_i$, hangs off $ST(\mathcal{R})$, then all nodes from $par(v_i)$ to the

root are in $ST(\mathcal{R})$. This means there can be no other node hanging off $ST(\mathcal{R})$ on the path from $v_i$ to the root, as it would require a node not in $ST(\mathcal{R})$ on this path. Neither can any node descended from $v_i$ hang off $ST(\mathcal{R})$. If there were such a node, then all the ancestors of that node, including $v_i$, would be in $ST(\mathcal{R})$. Since $v_i$ hangs off $ST(\mathcal{R})$, it is not in $ST(\mathcal{R})$. Therefore, there can be at most one node hanging off $ST(\mathcal{R})$ on the path from any leaf to the root (exactly one if that leaf is not in $\mathcal{R}$, and none if it is not in $\mathcal{R}$).

Suppose there is a leaf $v_l \in S_{i_1,j_1} \cap S_{i_2,j_2}$. So each of the Special Paths corresponding to $S_{i_1,j_1}$ and $S_{i_2,j_2}$ has a node hanging off that is an ancestor of $v_l$. But there can be only one node hanging off $ST(\mathcal{R})$ on the path from $v_l$ to the root, so this must be the same node for both Special Paths. So these two paths, $v_{i_1}, \ldots, par(v_{j_1})$ and $v_{i_2}, \ldots, par(v_{j_2})$, have a node in common. This contradicts Lemma 67, which says any node in $ST(\mathcal{R})$ can only belong to at most one Special Path. Therefore there is no $v_l \in S_{i_1,j_1} \cap S_{i_2,j_2}$ and $S_{i_1,j_1} \cap S_{i_2,j_2} = \emptyset$. □

It is important that we establish that any cover of the Subset Difference Revocation Scheme is a disjoint cover. The following Lemma shows that the union of two subsets of the Subset Difference Revocation Scheme which are not disjoint, is itself a subset of the Subset Difference Revocation Scheme. In the proof, whenever we say two nodes on a binary tree are *directly related*, we mean that one is a descendant of, or equal to the other. We also use the fact that:

$$\gamma(i,j) = desc(i) \setminus desc(j)$$

$$\text{which implies} \qquad \gamma(i,j) \subset desc(i).$$

From Lemma 20, we know that if the sets of descendants of two nodes in a binary tree intersect, then one is contained in the other. Specifically, if $j$ is descended from $i$ (or if $j = i$) then $desc(j) \subseteq desc(i)$. If $i$ and $j$ are not related then $desc(i) \cap desc(j) = \emptyset$.

In the proof, we consider two subsets in the Subset Difference Revocation Scheme: $\gamma(i_1, j_1)$ and $\gamma(i_2, j_2)$. We will look at all the possibilities for the different relative positions of the four nodes $i_1, j_1, i_2$ and $j_2$. For each possibility, we need to show either $\gamma(i_1, j_1) \cap \gamma(i_2, j_2) = \emptyset$ or $\gamma(i_1, j_1) \cup \gamma(i_2, j_2) = \gamma(i, j)$,

where $\gamma(i,j)$ is some other subset in the Subset Difference Revocation Scheme. The order we look at the different possibilities is:

1. $desc(i_1) \cap desc(i_2) = \emptyset$
2. $desc(i_1) = desc(i_2)$
3. $desc(i_1) \supset desc(i_2)$
   (a) $desc(i_2) \subseteq desc(j_1)$
   (b) $desc(i_2) \cap desc(j_1) = \emptyset$
   (c) $desc(i_2) \supset desc(j_1)$
      i. $desc(j_1) \supseteq desc(j_2)$
      ii. $desc(j_1) \subseteq desc(j_2)$
      iii. $desc(j_1) \cap desc(j_2) = \emptyset$
4. $desc(i_1) \subset desc(i_2)$

*Lemma* 70. Let $SDRS = (\mathcal{N}, \Omega, \gamma)$ be a Subset Difference Revocation Scheme. For any $(i_1,j_1),(i_2,j_2) \in \Omega$, $\gamma(i_1,j_1) \cap \gamma(i_2,j_2) \neq \emptyset$ implies $\gamma(i_1,j_1) \cup \gamma(i_2,j_2) = \gamma(i,j)$, for some $(i,j) \in \Omega$.

*Proof.* Let us first consider the case when $i_1$ and $i_2$ are not directly related. No path from $i_1$ to a leaf goes through $i_2$ and vice versa. This means that $desc(i_1) \cap desc(i_2) = \emptyset$. Therefore, $\gamma(i_1,j_1) \cap \gamma(i_2,j_2) = \emptyset$, as both subsets are contained in the respective set of descendants, i.e. we have $\gamma(i_1,j_1) \subset desc(i_1)$ and $\gamma(i_2,j_2) \subset desc(i_2)$. So if $\gamma(i_1,j_1)$ and $\gamma(i_2,j_2)$ intersect, then either $i_1 = i_2$, $i_1$ is a descendant of $i_2$ or $i_2$ is a descendant of $i_1$.

Suppose $i_1 = i_2$. If the two nodes $j_1$ and $j_2$ are also equal, then the subsets are the same ($\gamma(i_1,j_1) = \gamma(i_2,j_2)$). If $j_1$ is a descendant of $j_2$ then $desc(j_1) \subseteq desc(j_2)$. If we let $i = i_1 = i_2$, then we get:

$$\begin{aligned} \gamma(i_2,j_2) = \gamma(i,j_2) &= desc(i) \setminus desc(j_2) \\ &\subseteq desc(i) \setminus desc(j_1) \\ &= \gamma(i,j_1) = \gamma(i_1,j_1). \end{aligned}$$

Hence, with $j = j_1$, we have $\gamma(i_1, j_1) \cup \gamma(i_2, j_2) = \gamma(i, j)$. The same argument can be made if $j_2$ is the descendant of $j_1$, the only difference is that $\gamma(i_1, j_1) \cup \gamma(i_2, j_2) = \gamma(i, j)$, with $j = j_2$. If $j_1$ is not a descendant of $j_2$, and $j_2$ and is not a descendant of $j_1$ then $desc(j_1) \cap desc(j_2) = \emptyset$. Hence:

$$\begin{aligned}\gamma(i_1, j_1) \cup \gamma(i_2, j_2) &= (desc(i_1) \setminus desc(j_1)) \cup (desc(i_2) \setminus desc(j_2)) \\ &= desc(i) \setminus (desc(j_1) \cap desc(j_2)) \\ &= desc(i).\end{aligned}$$

If $i$ is the root then $\gamma(i_1, j_1) \cup \gamma(i_2, j_2) = \gamma(0, 0)$, otherwise we can write the union as $\gamma(i_1, j_1) \cup \gamma(i_2, j_2) = \gamma(par(i), sib(i))$. Thus the result holds when $i_1 = i_2$.

Suppose now that $i_2$ is a descendant of $i_1$. If $j_1 = i_2$, or $i_2$ is a descendant of $j_1$ then $desc(i_2) \subseteq desc(j_1)$. Since:

$$\begin{aligned}\gamma(i_1, j_1) = desc(i_1) \setminus desc(j_1) \qquad &\text{and} \qquad desc(i_2) \subseteq desc(j_1) \\ \text{we have} \quad \gamma(i_1, j_1) \cap desc(i_2) &= \emptyset \\ \text{So} \quad \gamma(i_1, j_1) \cap \gamma(i_2, j_2) &= \emptyset.\end{aligned}$$

Hence, we may suppose that $j_1$ is a descendant of $i_2$ or $j_1$ and $i_2$ are not directly related. If the latter is the case, then $\gamma(i_2, j_2) \subseteq desc(i_2) \subseteq \gamma(i_1, j_1)$. So, $\gamma(i_1, j_1) \cup \gamma(i_2, j_2) = \gamma(i_1, j_1)$.

We are left with the option of $j_1$ being a descendant of $i_2$. If $j_1$ and $j_2$ are related (either they are the same or one is descended from the other), then we set $j$ to be the node furthest from the root. That way:

$$\begin{aligned}desc(j_1) \cap desc(j_2) &= desc(j) \\ \text{so that} \quad \gamma(i_1, j_1) \cup \gamma(i_2, j_2) &= (desc(i_1) \setminus desc(j_1)) \cup (desc(i_2) \setminus desc(j_2)) \\ &= (desc(i_1) \cup desc(i_2)) \setminus (desc(j_1) \cap desc(j_2)) \\ &= desc(i_1) \setminus desc(j) = \gamma(i_1, j).\end{aligned}$$

If $j_1$ and $j_2$ are not related then $desc(j_2) \subset \gamma(i_1, j_1)$ since $j_2 \in desc(i_1)$ but $j_2 \notin desc(j_1)$. Similarly, $desc(j_1) \subset \gamma(i_1, j_1)$ since $j_1 \in desc(i_2)$ but $j_1 \notin desc(j_2)$. Therefore:

$$\gamma(i_1, j_1) \cup \gamma(i_2, j_2) = \begin{cases} \gamma(0, 0)) & \text{if } i \text{ is the root} \\ \gamma(par(i), sib(i)) & \text{otherwise} \end{cases}.$$

Therefore, the result is true when $i_2$ is a descendant of $i_1$. Because $i_1$ and $i_2$ are just any nodes in the binary tree, without loss of generality, the result is also true when $i_1$ is a descendant of $i_2$. □

*Corollary* 71. Let $SDRS$ be a Subset Difference Revocation Scheme. Any minimal cover of $\mathcal{N} \setminus \mathcal{R}$ in $SDRS$ is disjoint.

*Proof.* If there is any pair of subsets in the cover of $\mathcal{N} \setminus \mathcal{R}$ with a non-trivial intersection, then by Lemma 70, there exists one subset in the scheme that is equal to their union. Therefore, no minimal cover can have intersecting subsets, i.e. a minimal cover must be disjoint. □

Finally, we show that each subset in a minimal cover corresponds to a Special Path in $ST(\mathcal{R})$, and also there can be no Special Path in $ST(\mathcal{R})$ that does not correspond to one of the subsets.

*Lemma* 72. Let $RS = (\mathcal{N}, \Omega, \gamma)$ be a Subset Difference Revocation Scheme defined on the complete binary tree $T$. Then for any $\mathcal{R} \subseteq \mathcal{N}$, $t(\mathcal{N}, \mathcal{R})$ is equal to the number of Special Paths on the tree.

*Proof.* Let $\gamma(i_1, j_1), \ldots, \gamma(i_\alpha, j_\alpha)$ be a minimal cover of $\mathcal{N} \setminus \mathcal{R}$. By Lemma 68, we know that for any Special Path $S_{i,j}$, $\gamma(i, j) = \{f(v_l) : v_l \in S_{i,j}\}$. But we need to show that the paths corresponding to the subsets in the cover are Special Paths.

Suppose that for one of the sets in the cover, $\gamma(i_1, j_1)$, the path $S_{i_1,j_1}$ is not a Special Path. Since $\gamma(i_1, j_1)$ is in the cover, each node on the path from $v_{i_1}$ to $par(v_{j_1})$ has one child not in $ST(\mathcal{R})$. If $S_{i_1,j_1}$ is not a Special Path then either $v_{j_1}$ is not a leaf nor a node of degree 3, or else the parent of $v_{i_1}$ has a child not in $ST(\mathcal{R})$. In the former case, $v_{j_1}$ must have a node hanging off. In order for the descendant leaves to be covered, there must be another subset in the cover, $\gamma(i_2, j_2)$, where $i_2$ is an ancestor of (or the same node as) $j_1$. If $j_1$ and $j_2$ are the children of $i_2$, then all nodes descended from $i_1$ are privileged and can be covered with one index instead of two (this contradicts the assumption that the cover was minimal). Otherwise, if $i_2$ is an ancestor of $j_1$, then there will be an intersection of the two subsets $\gamma(i_1, j_1)$ and $\gamma(i_2, j_2)$,

which contradicts Corollary 71. The only option left is that $j_1 = i_2$. By the nature of $\gamma$, we have:

$$\begin{aligned}\gamma(i_1, j_1) \cup \gamma(i_2, j_2) &= \{desc(i_1) \setminus desc(j_1)\} \cup \{desc(i_2) \setminus desc(j_2)\} \\ &= \{desc(i_1) \setminus desc(j_1)\} \cup \{desc(j_1) \setminus desc(j_2)\} \\ &= desc(i_1) \setminus desc(j_2) = \gamma(i_1, j_2).\end{aligned}$$

This also contradicts the assumption that the cover was minimal.

So, if $S_{i_1,j_1}$ is not a Special Path then we must have that the parent of $v_{i_1}$ has a node hanging off. The above argument still applies, this time the extra node hanging off occurs above $i_1$, rather than below $j_1$. There must be some subset in the cover, $\gamma(i_2, j_2)$, that includes the leaves descended from the extra node hanging off. We must have $i_2$ an ancestor of $i_1$ to cover these leaves, and $j_2 = i_1$ for the same reasons as in the earlier case. Therefore, we get:

$$\begin{aligned}\gamma(i_1, j_1) \cup \gamma(i_2, j_2) &= \{desc(i_1) \setminus desc(j_1)\} \cup \{desc(i_2) \setminus desc(j_2)\} \\ &= \{desc(j_2) \setminus desc(j_1)\} \cup \{desc(i_1) \setminus desc(j_2)\} \\ &= desc(i_2) \setminus desc(j_1) = \gamma(i_2, j_1),\end{aligned}$$

which again contradicts the assumption that the cover was minimal.

This proves that each subset in a minimal cover corresponds to a Special Path in $ST(\mathcal{R})$. But to prove that $t(\mathcal{N}, \mathcal{R})$ equals the number of Special Paths, we must prove that there can be no more Special Paths in $ST(\mathcal{R})$. Suppose we had a Special Path in $ST(\mathcal{R})$, $S_{i_{\alpha+1},j_{\alpha+1}}$, distinct from the $\alpha$ Special Paths, $S_{i_1,j_1}, \ldots, S_{i_\alpha,j_\alpha}$ for the $\alpha$ subsets in the cover. By the definition of a Special Path, the leaves in $S_{i_{\alpha+1},j_{\alpha+1}}$ are privileged. None of these leaves can be descended from any of the $\alpha$ Special Paths, since these sets are disjoint (Lemma 69). So there are privileged leaves not contained in the cover, and so $S_{i_1,j_1}, \ldots, S_{i_\alpha,j_\alpha}$ (which equals $\gamma(i_1, j_1), \ldots, \gamma(i_\alpha, j_\alpha)$) is not a cover. This contradictions the assumption on the $\alpha$ subsets. Therefore:

$$t(\mathcal{N}, \mathcal{R}) = |\{S_{i,j} : S_{i,j} \text{ is a Special Path on } ST(\mathcal{R})\}|. \qquad \square$$

This means that the size of the minimal cover, $t(\mathcal{N}, \mathcal{R})$, is the number of maximal paths of nodes in $ST(\mathcal{R})$ where each node has one node hanging off

that is not in $ST(\mathcal{R})$. To find $\mathcal{R}$, with $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ and $|\mathcal{R}| = r$, we want to choose $\mathcal{R}$ so that $ST(\mathcal{R})$ has as many Special Paths as possible. This suggests that these paths be as short as possible. The shortest Special Path are in the form $S_{i,j}$, where $v_i$ is $v_j$'s parent, i.e. length 1. Also, we need to minimise the number of null Special Paths, i.e. those with no nodes not in $ST(\mathcal{R})$ hanging off. These do not increase $t(\mathcal{N}, \mathcal{R})$ at all and can be in the form of two nodes of degree 3, where one is the parent of the other, a node of degree 3 being the parent of two leaves in $ST(\mathcal{R})$, or the root having degree 2 (the root can be the first node in a Special Path, provided it has only one child in $ST(\mathcal{R})$). We formalise these ideas in the following Lemma. We show that the combined number of Special Paths and null Special Paths is always constant for a given $r$. This is a useful result since the size of a minimal cover is just the number of Special Paths in $ST(\mathcal{R})$.

*Lemma* 73. Let $\mathcal{R}$ be a non-empty subset of leaves of a complete binary tree $T$, $|\mathcal{R}| = r$. Then the sum of the number of Special Paths in $ST(\mathcal{R})$ and the number of null Special Paths in $ST(\mathcal{R})$ is $2r - 1$.

*Proof.* By Lemma 10 we have that the number of nodes in $ST(\mathcal{R})$ with both children in $ST(\mathcal{R})$ is $|\mathcal{R}| - 1 = r - 1$. Obviously, the number of leaves in $ST(\mathcal{R})$ is $|\mathcal{R}| = r$. Since there is no overlap between the two types of nodes (leaves do not have children), the set $S$ defined to be the set of all nodes with both children in $ST(\mathcal{R})$ and all leaves has $|S| = 2r - 1$.

From Lemma 67 we know that all nodes in $ST(\mathcal{R})$ are partitioned into Special Paths and null Special Paths. So each node of $S$ belongs to a Special Path or is a null Special Path. Since null Special Paths are just individual nodes on the subtree with no node hanging off, distinct null Special Paths will be distinct nodes in $S$. Furthermore, each Special Path contains exactly one node in $S$. The furthest node from the root in a Special Path ($v_j$) is either a leaf or a node with both children in $ST(\mathcal{R})$. Therefore, there is a one-to-one correspondence between nodes in $S$ and the set of Special Paths and null Special Paths. So the total number of Special Paths in $ST(\mathcal{R})$ and the total number of null Special Paths in $ST(\mathcal{R})$ add up to the size of $S$ which equals $2r - 1$. □

The implication for the Subset Difference Revocation Scheme follows directly:

*Corollary* 74. Let $RS = (\mathcal{N}, \Omega, \gamma)$ be a Subset Difference Revocation Scheme defined on the complete binary tree $T$. Then for any $\mathcal{R} \subseteq \mathcal{N}$, $\mathcal{R} \neq \emptyset$:

$$t(\mathcal{N}, \mathcal{R}) = 2r - 1 - \#\text{ null Special Paths in } ST(\mathcal{R}).$$

*Proof.* By Lemma 72, $t(\mathcal{N}, \mathcal{R})$ is the number of Special Paths in $ST(\mathcal{R})$. By Lemma 73, $2r-1$ equals the number of Special Paths and null Special Paths. Therefore, $t(\mathcal{N}, \mathcal{R}) = 2r - 1 -$ number of null Special Paths in $ST(\mathcal{R})$. □

### 6.1.1 Special Subtrees

Corollary 74 gives us a very obvious way to describe a subset $\mathcal{R}$ that gives $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$, namely a subset such that $ST(\mathcal{R})$ has the minimum number of null Special Paths. If ever we have a subtree $ST(\mathcal{R})$ with no null Special Paths, then we clearly have $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$:

*Definition* 75. Let $\mathcal{R}$ be a non-empty subset of leaves of a complete binary tree $T$. If $ST(\mathcal{R})$ has no null Special Paths, then we call $ST(\mathcal{R})$ a *Special Subtree*. If a Special Subtree on $T$ contains $r$ leaves, and there exists no Special Subtree on $T$ with greater than $r$ leaves, then we call the Special Subtree a *Maximal Special Subtree*.

One immediate property of Special Subtrees is that they have $2r-1$ Special Paths (Lemma 73). So the bound of Naor of $t_{\max}(n, r) \leq 2r - 1$ is only tight when $ST(\mathcal{R})$ is a Special Subtree. We will use Special Subtrees to find the range for which Naor's bound is tight. We do this by first working out the number of leaves in a Maximal Special Subtree, which is also the greatest value of $r$ such that $t_{\max}(n, r) = 2r - 1$. We then observe that any subset of leaves of a Maximal Special Subtree define a Special Subtree, proving the result for all lower (positive) $r$. Also, Special Subtrees will be used to find $ST(\mathcal{R})$ such that $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ for values of $r$ outside this range.

*Lemma* 76. Let $\mathcal{R}$ be a non-empty subset of leaves of a complete binary tree $T$ of height $h$. If $ST(\mathcal{R})$ is a Maximal Special Subtree, then:

$$|\mathcal{R}| = 2^{\lfloor \frac{h-1}{2} \rfloor}.$$

*Proof.* Since $ST(\mathcal{R})$ is a Maximal Special Subtree, $ST(\mathcal{R})$ has no null Special Paths. By the definition of a null Special Path, this means that all the following must be true of $ST(\mathcal{R})$:

1. The root has degree 1 in $ST(\mathcal{R})$
2. If $v_j$ is an internal node of degree 3, then $par(v_j)$ is the root, or a node of degree 2
3. All leaves in $ST(\mathcal{R})$ have a parent of degree 2 (or degree 1 if it is the root).

Since the root has degree 1, all $r$ leaves are descended from one of the children of the root, which is at height $h-1$. The second condition means that all degree 3 nodes are separated by a path of length at least 2. By the third condition, the same is true of degree 3 nodes and leaves. By the exact same argument as Lemma 9, any such binary tree with $r$ leaves must have height $2\lceil \log_2(r) \rceil$. Since all $r$ leaves are descended from a node at height $h-1$, we have:

$$\begin{aligned} & 2\lceil \log_2(r) \rceil \leq h-1 \\ \text{So that} \qquad & \lceil \log_2(r) \rceil \leq \frac{h-1}{2} \\ \text{i.e.} \qquad & \log_2(r) \leq \left\lfloor \frac{h-1}{2} \right\rfloor \\ \text{which implies} \qquad & r \leq 2^{\lfloor \frac{h-1}{2} \rfloor}. \end{aligned}$$

To prove the result, we will construct a Special Subtree with this many leaves. Because this meets the bound on $r$, there can be no more leaves and so the Special Subtree is Maximal.

Construct $ST(\mathcal{R})$ by combining the following paths from the root:

1. Start with a single path from the root to the left child of the root if $h$ is odd, and the left-most grandchild of the root if $h$ is even (call this node $v$).
2. From $v$, add two paths: one to the left grand-child of $v$, one to the left child of the right child of $v$.

3. From both of these end points, add two similar paths (one to the left grandchild, one to the left child of the right child).

4. Continue adding such pairs of paths until the only endpoints of $ST(\mathcal{R})$ left are leaves.

Clearly, this gives rise to an $ST(\mathcal{R})$ where the root has degree 1. Since all nodes of degree 3 have two nodes of degree 2 as children, we never have a pair of nodes $v_j,\ par(v_j)$ where both nodes have degree 3. And because $v$ is chosen to be at an even distance from the leaves, and each of the paths added are length 2, the parents of the leaves will be degree 2. So there are no null Special Paths in $ST(\mathcal{R})$ and $ST(\mathcal{R})$ is a Special Subtree.

The number of leaves in this tree can be calculated from the number of degree 3 nodes along any path (same for each path). This is $\frac{h-1}{2} = \lfloor \frac{h-1}{2} \rfloor$ if $h$ is odd, $\frac{h-2}{2} = \lfloor \frac{h-1}{2} \rfloor$ if $h$ is even. By the nature of the construction of $ST(\mathcal{R})$, if one path from the root to a leaf has a degree 3 node at a certain level, then all paths in $ST(\mathcal{R})$ have a degree 3 node at that level. Since one degree 3 node means the next level down will have one more node in $ST(\mathcal{R})$ than the current level, all nodes being degree 3 means the next level will have twice the number of nodes in $ST(\mathcal{R})$. Therefore the number of leaves is $|\mathcal{R}| = 2^{\lfloor \frac{h-1}{2} \rfloor}$. Since there can be no Special Subtree with more than $2^{\lfloor \frac{h-1}{2} \rfloor}$ leaves, $ST(\mathcal{R})$ is a Maximal Special Subtree. □

The subtree $ST(\mathcal{R})$ described in Lemma 76 will have the appearance of a complete tree in that in certain levels of the tree, all nodes at that level will have two children. However, these levels alternate with levels where each node only has one child (in a complete tree, all internal nodes have two children). Examples of these types of subtrees can be found in Figure 6.2. This type of subtree is not only important in determining $t_{\max}(n, r)$ for the range of $r$ in Lemma 78, but for all $r$. The following Lemma allows us to obtain Special Subtrees from subsets of leaves of any existing Special Subtree (including Maximal Special Subtrees).

*Lemma* 77. Let $\mathcal{R}$ be a subset of leaves of a complete binary tree $T$ such that $ST(\mathcal{R})$ is a Special Subtree. For any non-empty subset $\mathcal{R}' \subseteq \mathcal{R}$, $ST(\mathcal{R}')$ is also a Special Subtree.

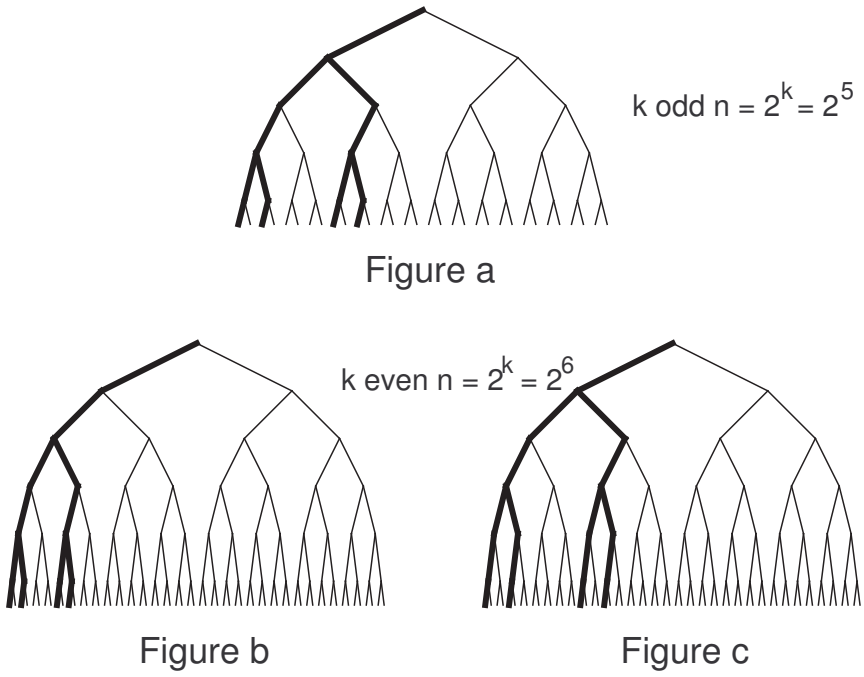


Figure 6.2: Maximum number of Special Paths in $ST(\mathcal{R})$

*Proof.* Let $ST(\mathcal{R})$ be a Special Subtree and let $\mathcal{R}' \subseteq \mathcal{R}$. Suppose $ST(\mathcal{R}')$ is not a Special Subtree. Since $\mathcal{R}' \subseteq \mathcal{R}$, that means $ST(\mathcal{R}')$ is a subtree of $ST(\mathcal{R})$. So the degree of any node in $ST(\mathcal{R}')$ is less than or equal to the degree of the same node in $ST(\mathcal{R})$. Conversely, any node in $ST(\mathcal{R})$ must have degree greater than or equal to the degree of the node in $ST(\mathcal{R}')$. Since $ST(\mathcal{R}')$ is not a Special Subtree, we have either the root with degree 2, a parent of a leaf with degree 3, or a parent of a node of degree 3 with degree 3 itself. Since the degree of the nodes in $ST(\mathcal{R})$ have to be greater than or equal than those in $ST(\mathcal{R}')$ we would also have a null Special Path in $ST(\mathcal{R})$. This contradicts the assumption on $ST(\mathcal{R})$. Therefore, $ST(\mathcal{R}')$ is a Special Subtree. □

Corollary 74 said that we must have $t_{\max}(n, r) = 2r - 1$ when there are no Special Paths in $ST(\mathcal{R})$, i.e. $ST(\mathcal{R})$ is a Special Subtree. Using both Lemmas 76 and 77 we can give the range of $r$ for when this happens.

*Lemma* 78. Let $SDRS$ be a Subset Difference Revocation Scheme with $n = 2^k$ users. Then $SDRS$ has:

$$t_{\max}(n, r) = 2r - 1,$$

if and only if $1 \leq r \leq 2^{\lfloor \frac{k-1}{2} \rfloor}$.

*Proof.* We know $t_{\max}(n, 0) = 1$, so obviously $t_{\max}(n, r) \neq 2r - 1$ for $r = 0$. By Corollary 17 (and also Corollary 74), $t_{\max}(n, r) \leq 2r - 1$ for all $r > 0$.

A Special Subtree is defined to be a Steiner Tree with $2r-1$ Special Paths. Since $t(\mathcal{N}, \mathcal{R})$ is the number of Special Paths in $ST(\mathcal{R})$, $t(\mathcal{N}, \mathcal{R}) = 2r - 1$ if and only if there exists a subset $\mathcal{R}$, with $|\mathcal{R}| = r$, such that $ST(\mathcal{R})$ is a Special Subtree. By Lemma 76, the most leaves that can be in a Special Subtree is $2^{\lfloor \frac{k-1}{2} \rfloor}$ (there are $n = 2^k$ users, so the complete binary tree has height $k$). This means that $t_{\max}(n, r) < 2r-1$ for $r > 2^{\lfloor \frac{k-1}{2} \rfloor}$. Also from Lemma 76, there exists a Maximal Subtree with $r = 2^{\lfloor \frac{k-1}{2} \rfloor}$ leaves. A Maximal Special Subtree has the property of a Special Subtree that it has $2r - 1$ Special Paths. Therefore $t_{\max}(n, r) = 2r - 1$ for $r = 2^{\lfloor \frac{k-1}{2} \rfloor}$. By Lemma 77, for any $1 \leq r < 2^{\lfloor \frac{k-1}{2} \rfloor}$, we can choose any $r$ leaves of the Maximal Special Subtree and get a Special Subtree. This gives $t_{\max}(n, r) = 2r - 1$ for $1 \leq r \leq 2^{\lfloor \frac{k-1}{2} \rfloor}$. □

Figure 6.2 shows examples of the construction in Lemma 76, both when $k$ is odd and even. There are many different possible $ST(\mathcal{R})$, where $\mathcal{R}$ has $t(\mathcal{N}, \mathcal{R}) = 2|\mathcal{R}| - 1$, and the above construction can be modified to generate them. Whenever we have a degree 2 node, either the left child or the right child can be in $ST(\mathcal{R})$. This gives us several different choices of $\mathcal{R}$ that have the same sized cover. But whenever $k$ is even, there is a more important degree of freedom. In the construction, the degree 3 node closest to the root is distance 2 away. Because of this, the output of the construction, as shown in Figure 6.2b, has a length 2 Special Path at the root of the tree. This is 1 edge longer than the shortest possible Special Path, all other Special Paths in $ST(\mathcal{R})$ have length 1. This length 2 Special Path has to appear somewhere in the tree, but shifting it lower down the tree does not change the size of the cover. In Figure 6.2c we have the length 2 Special Paths terminated by the leaves. It is this extra node which allows us to extend the result of Lemma 78.

*Corollary* 79. Let $SDRS$ be a Subset Difference Revocation Scheme with $n = 2^k$ users, $k$ even. Then $SDRS$ has:

$$t_{\max}(n, r) = 2r - 2,$$

if $2^{\lfloor \frac{k-1}{2} \rfloor} < r \leq 2^{\lfloor \frac{k-1}{2} \rfloor + 1}$.

*Proof.* By Lemma 78 we have that $t_{\max}(n,r) < 2r-1$ for all $r > 2^{\lfloor \frac{k-1}{2} \rfloor}$, or $t_{\max}(n,r) \leq 2r-2$ as the size of any cover must be a whole number. Since the binary tree has even height, the two children of the root are at an odd height. Define $ST(\mathcal{R})$ to be comprised of two of the Maximal Special Subtrees (as constructed in Lemma 76) rooted at the two children of the root, as well as the two edges from these children to the root. $\mathcal{R}$ is the set of leaves of these Maximal Special Subtrees. Since each Maximal Special Subtree has $2^{\lfloor \frac{k-2}{2} \rfloor}$ leaves:

$$|\mathcal{R}| = 2.2^{\lfloor \frac{k-2}{2} \rfloor} = 2^{\lfloor \frac{k-2}{2} \rfloor + 1} = 2^{\lfloor \frac{k-1}{2} \rfloor + 1}, \quad \text{since } k \text{ is even.}$$

By Lemma 67, all nodes in $ST(\mathcal{R})$ are partitioned into Special Paths and null Special Paths. The root of $ST(\mathcal{R})$ is a null Special Path as both its children are in $ST(\mathcal{R})$. Since $ST(\mathcal{R})$ from both children of the root down are Maximal Special Subtrees, the rest of $ST(\mathcal{R})$ is just Special Paths. The number of Special Paths in any Special Subtree is $2r'-1$, where $r'$ is the number of leaves in that Special Subtree. Since there were $2^{\lfloor \frac{k-2}{2} \rfloor}$ leaves in each Maximal Special Subtree, the total number of Special Paths in $ST(\mathcal{R})$ is:

$$2.2^{\lfloor \frac{k-2}{2} \rfloor} - 1 + 2.2^{\lfloor \frac{k-2}{2} \rfloor} - 1 = 2.2^{\lfloor \frac{k-1}{2} \rfloor + 1} - 2.$$

So for $r = 2^{\lfloor \frac{k-1}{2} \rfloor + 1}$, we have $t_{\max}(n,r) = 2r-2$. For any $2^{\lfloor \frac{k-1}{2} \rfloor} < r < 2^{\lfloor \frac{k-1}{2} \rfloor + 1}$, we can remove any $2^{\lfloor \frac{k-1}{2} \rfloor + 1} - r$ leaves from either Maximal Special Subtree. By Lemma 77, and the fact that we are only removing $2^{\lfloor \frac{k-1}{2} \rfloor + 1} - r < 2^{\lfloor \frac{k-1}{2} \rfloor}$ leaves, we will still have two Special Subtrees rooted at the children of the root. So the number of Special Paths is still $2r-2$. □

So we know what $t_{\max}(n,r)$ looks like for very small $r$. It is also simple to describe $t_{\max}(n,r)$ for $r = n/2$. When we looked at the Complete Subtree Revocation Scheme, the maximum $t(\mathcal{N},\mathcal{R})$ over all $\mathcal{R} \subseteq \mathcal{N}$ occurred when every second user was revoked. If $\mathcal{R}$ is comprised of every second user as they appear in the binary tree, then all nodes in the level above the leaves have degree 2 in $ST(\mathcal{R})$. Since all nodes in this level are in $ST(\mathcal{R})$, no node in any higher level will have a node not in $ST(\mathcal{R})$ hanging off. We have $n/2$ nodes in the level above the leaves. Each node is degree 2 in $ST(\mathcal{R})$, and the parent of each node is a node of degree 3. This gives us $n/2$ Special Paths.

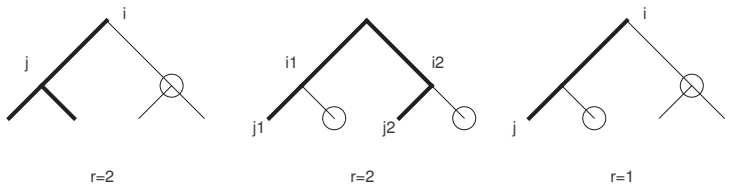


Figure 6.3: Examples of $ST(\mathcal{R})$ two levels above the leaves.

Therefore $t(\mathcal{N}, \mathcal{R}) = n/2$. For any Revocation Scheme we have $t_{\max}(n, r) \leq n - r$ (Corollary 6), so $t_{\max}(n, n/2) = n/2$. We can extend this to $n/2 < r \leq n$. The union of $\mathcal{R}$ and any other $r - n/2$ leaves, gives $\mathcal{R}'$ with $|\mathcal{R}'| = r$ and $n/2 - (r - n/2)$ nodes of degree 2 in $ST(\mathcal{R})'$. This gives $n - r$ Special Paths, and so $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r) = n - r$ for $n/2 \leq r \leq n$, as was the case for the Complete Subtree Revocation Scheme.

One immediate difference between Complete Subtree and Subset Difference is that the former cannot cover 3 privileged users in a tree of 4 with one subset, while the latter can. In the third diagram in Figure 6.3 we see an example of this. This means that the choice of $\mathcal{R}$ that gave $t_{\max}(n, r) = n/2$ for $n/4 \leq r \leq n/2$ with the Complete Subtree Revocation Scheme, will not do the same for the Subset Difference Revocation Scheme. In fact, a consequence of the formula for $t_{\max}(n, r)$ that we will derive is that $r = n/2$ is the only point where $t_{\max}(n, r) = n/2$.

## 6.2 Maximising $t(\mathcal{N}, \mathcal{R})$ over all $M(\mathcal{R})$

We are now going to define a function on subsets of $\mathcal{N}$, that on input of $\mathcal{R}$ will output a subset, $\mathcal{R}'$, with the same number of leaves, but with $ST(\mathcal{R}')$ different from $ST(\mathcal{R})$ in such a way that $t(\mathcal{N}, \mathcal{R}')$ cannot be less than $t(\mathcal{N}, \mathcal{R})$. As the function will be defined for all subsets of $\mathcal{N}$, it can be applied to those subsets that give $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. This will allow us to classify some (but not all) subsets $\mathcal{R}$ that give the maximum $t(\mathcal{N}, \mathcal{R})$. The covers of such $\mathcal{N} \setminus \mathcal{R}$ will have properties derived from the function that will make it easier to count the size of the cover (and so giving us a formula for $t_{\max}(n, r)$).

*Definition* 80. Let $T$ be a complete binary tree with $2n-1$ nodes $\{v_1, \ldots, v_{2n-1}\}$,

| | **Definition of** $M(\mathcal{R})$ |
|---|---|
| | **Initialise:** $\mathcal{R}' = \mathcal{R}$ |
| 0: | **Let** $S$ be the set of triples of nodes in $ST(\mathcal{R}')$, $(v_i, v_j, v_k)$, such that: $(v_i, v_j)$ is a Special Path with $v_j$ degree 3 $v_k$ is the child of $v_j$ and is either a leaf or a node of degree 3 |
| 1: | **while** $S \neq \emptyset$ **do** |
| 2: | **Let** $(v_i, v_j, v_k) \in S$ |
| 3: | Add leaves to $\mathcal{R}'$ such that $ST(\mathcal{R}')$ from the left child of $v_j$'s sibling down is the same as $ST(\mathcal{R}')$ from $v_k$ down |
| 4: | Remove those descendants of $v_k$ in $\mathcal{R}'$ from $\mathcal{R}'$ |
| 5: | Update $S$ |
| 6: | **end do** |

Table 6.1: Definition of function $M(\mathcal{R})$.

indexed using breadth first labelling. Let $\mathcal{N} = \{v_n, \ldots, v_{2n-1}\}$ be the set of leaves of $T$. The function $M : 2^{\mathcal{N}} \to 2^{\mathcal{N}}$ for input $\mathcal{R} \subseteq \mathcal{N}$ is defined in Table 6.1. The resulting subset $\mathcal{R}'$ is the output $M(\mathcal{R})$ of $M$. Any subset in the range of $M$ is an $M$-type subset, i.e. $\mathcal{R}'$ is an $M$-type subset if there exists a subset $\mathcal{R}$ with $M(\mathcal{R}) = \mathcal{R}'$.

We can give a more descriptive explanation of what $M(\mathcal{R})$ does. The only types of nodes that can terminate a Special Path are a leaf, the root or a node of degree 3. If we consider two Special Paths on the same path from any leaf to the root, then neither the root nor a leaf can occur between them. Provided that no other Special Path exists between these two paths, then they must be separated by at least one node of degree 3. The purpose of this function is to create a set $\mathcal{R}'$ such that all Special Paths on $ST(\mathcal{R}')$ on a path between the root and a leaf are separated by just one node of degree 3. For any input $\mathcal{R}$, if there are any instances of two (or more) such nodes separating two Special Paths, the Special Path higher up the tree is moved down by a rearrangement of the leaves.

In order for the function to be of use, $M(\mathcal{R})$ must be defined for all inputs $\mathcal{R}$. For this to be true, the algorithm must always terminate, and to prove this we will show that each pass through the while loop results in changes to the Steiner Tree that cannot be repeated indefinitely.

*Lemma* 81. Let $\mathcal{R}$ be any non-empty subset of leaves of a complete binary tree $T$. Let $\mathcal{R}'$ be the result of one pass through the while loop of the algorithm in calculating $M(\mathcal{R})$. Then $t(\mathcal{N}, \mathcal{R}') \geq t(\mathcal{N}, \mathcal{R})$. Furthermore, if $t(\mathcal{N}, \mathcal{R}') = t(\mathcal{N}, \mathcal{R})$ then all null Special Paths in $ST(\mathcal{R}')$ have a corresponding null Special Path in $ST(\mathcal{R})$ that is at the same height, with the exception of one null Special Path that is distance two closer to the root in $ST(\mathcal{R}')$ than in $ST(\mathcal{R})$.

*Proof.* We are interested in the difference in the numbers of (and positions of) the null Special Paths and Special Paths between $ST(\mathcal{R})$ and $ST(\mathcal{R}')$. Since these paths are defined by the degrees of the nodes, we need to specify the nodes that have different degrees in the two subtrees. Since the cental while loop is executed, we know that there must be 3 nodes $(v_i, v_j, v_k)$ in $ST(\mathcal{R})$ such that $v_i, \ldots, v_j$ is a Special Path, $v_k$ is a child of $v_j$ and either a leaf or a node of degree 3. The most obvious difference is $v_j$: it is chosen so that it is a degree 3 node in $ST(\mathcal{R})$, but one of its children $(v_k)$ has no descendants in $ST(\mathcal{R}')$, so it has degree 2 in $ST(\mathcal{R}')$. There is also a difference in the degrees of $par(v_j)$. This node is in a Special Path in $ST(\mathcal{R})$, and since it is not the node furthest from the root (which is $v_j$) it must have a node hanging off, and so it has degree 2 in $ST(\mathcal{R})$. However, both $v_j$ and its sibling have descendants in $\mathcal{R}'$, so $par(v_j)$ must have degree 3 in $ST(\mathcal{R}')$. Strictly speaking, all nodes descended from $v_k$ and the left child of $v_j$'s sibling have different degrees in the two trees. But because the portion of $ST(\mathcal{R})$ descended from $v_k$ is replicated below the left child of $v_j$'s sibling in $ST(\mathcal{R}')$, all null Special Paths and Special Paths are replicated as well. The portions of both trees descended from $v_k$ and from the right child of $v_j$'s sibling are exactly the same, as is the rest of the subtrees not descended from $v_i$.

We can now describe the difference in the Special Paths between the two subtrees. Since there are only three nodes with different degrees in the two trees, $par(v_j)$, $v_j$ and its sibling, we can consider two different portions of the subtrees independently: above $par(v_j)$, and below $v_j$ and $sib(v_j)$. Each node above $par(v_j)$ up to $v_i$ has a node hanging off $ST(\mathcal{R})$ (there is a Special Path from $v_i$ to $v_j$). But in $ST(\mathcal{R}')$, $v_i$ to $v_j$ is no longer a Special Path since $par(v_j)$ has degree 3. We will only have that $v_i$ to $par(v_j)$ is a Special Path in $ST(\mathcal{R}')$ if $v_i \neq par(v_j)$. Otherwise, we have a null Special Path at $v_i = par(v_j)$, because

$v_i = par(v_j)$ has degree 3 in $ST(\mathcal{R}')$ and $par(v_i)$ is unchanged from $ST(\mathcal{R})$ where did not have a node hanging off. So along this path one Special Path in $ST(\mathcal{R})$ can correspond to either a Special Path or a null Special Path in $ST(\mathcal{R}')$.

There are also two possibilities for the nodes below $v_j$ and $sib(v_j)$. In $ST(\mathcal{R})$, we know $v_j$ had degree 3 and $v_k$ either had degree 3 as well, or is a leaf (and so $v_k$ is a null Special Path in $ST(\mathcal{R})$). Since $par(v_j)$ is degree 2, $sib(v_j)$ is not in $ST(\mathcal{R})$ (neither are any of its descendants). The different cases occur depending on whether $sib(v_k)$ has a node not in $ST(\mathcal{R})$ hanging off or not. Suppose $sib(v_k)$ does have a node hanging off. Since $v_j = par(sib(v_k))$ had degree 3, $sib(v_k)$ is the highest node of a Special Path that terminates at some descendant node. In $ST(\mathcal{R}')$, $v_j$ is degree 2. Whereas in $ST(\mathcal{R})$ the highest node of the Special Path was $sib(v_k)$, in $ST(\mathcal{R}')$ it is $v_j$. The degree 3 node in $ST(\mathcal{R})$, $v_k$, is moved to the left child of $sib(v_j)$ in $ST(\mathcal{R}')$. Where this was a null Special Path in $ST(\mathcal{R})$, it is a Special Path in $ST(\mathcal{R}')$, since $sib(v_j)$ only has one child in $ST(\mathcal{R}')$. The net result is that one null Special Path and one Special Path in $ST(\mathcal{R})$ become two Special Paths in $ST(\mathcal{R}')$.

Alternatively, suppose $sib(v_k)$ does not have a node hanging off, either because it has degree 3 in $ST(\mathcal{R})$, or is a leaf. Either way, this node will be a null Special Path in $ST(\mathcal{R})$ (as is $v_k$). In $ST(\mathcal{R}')$, $v_j$ has degree 2, as does the parent of the degree 3 node, the left child of $sib(v_j)$ (where $v_k$ is moved to). This results in two Special Paths in $ST(\mathcal{R}')$, from $v_j$ to $sib(v_k)$ and from $sib(v_j)$ to the left child of $sib(v_j)$, where there were two null Special Paths in $ST(\mathcal{R})$.

We have considered all the nodes that have different degrees in $ST(\mathcal{R})$ and $ST(\mathcal{R}')$, and all possible differences in the Special Paths and null Special Paths. Because no other nodes have different degrees, all other Special Paths and null Special Paths are the same in both trees (or in the case of those descended from $v_k$, at the same height). In terms of numbers of Special Paths, above $par(v_j)$, $ST(\mathcal{R}')$ can either have one less, or the same number of Special Paths as $ST(\mathcal{R})$, and below this node $ST(\mathcal{R}')$ can have either 1 or 2 more Special Paths than $ST(\mathcal{R})$. So by Lemma 72, $t(\mathcal{N}, \mathcal{R}') \geq t(\mathcal{N}, \mathcal{R})$. If $t(\mathcal{N}, \mathcal{R}') = t(\mathcal{N}, \mathcal{R})$, then we must have that in $ST(\mathcal{R})$ $v_i$ is the parent of $v_j$ and $v_k$'s sibling has a

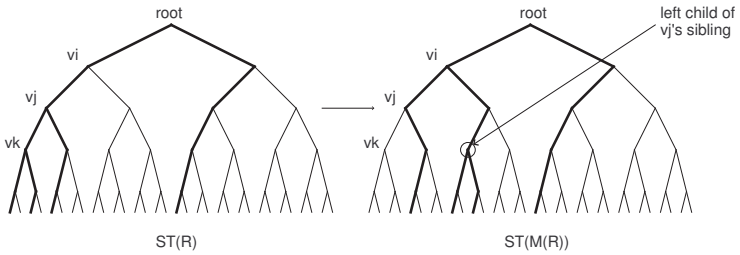


Figure 6.4: Example of $ST(M(\mathcal{R}))$

node hanging off $ST(\mathcal{R})$ ($v_k$ is the null Special Path). In $ST(\mathcal{R}')$ we have that $par(v_j)$ is the null Special Path, and both children of $par(v_j)$ are in Special Paths. So where there was a null Special Path at $v_k$ in $ST(\mathcal{R})$, there is one at $par(v_j)$ in $ST(\mathcal{R}')$. All other null Special Paths are at the same heights in both trees. Therefore, either $t(\mathcal{N}, \mathcal{R}') > t(\mathcal{N}, \mathcal{R})$ or there is one null Special Path distance two closer to the root in $ST(\mathcal{R}')$ than in $ST(\mathcal{R})$, the rest being at the same height. □

Figure 6.4 shows an example of the case when $t(\mathcal{N}, M(\mathcal{R})) = t(\mathcal{N}, \mathcal{R})$. There are five Special Paths and two null Special Paths in both $ST(\mathcal{R})$ and $ST(M(\mathcal{R}))$. The root is a null Special Path in both trees, since both children of the root are in both subtrees. The second null Special Path in $ST(\mathcal{R})$ is $v_k$, and in $ST(M(\mathcal{R}))$ is $v_i$, two edges closer to the root. Neither this change to the structure of $ST(\mathcal{R}')$, nor increasing the number of Special Paths in $ST(\mathcal{R}')$ can continue indefinitely, and so we get the following result.

*Corollary* 82. Let $\mathcal{R}$ be any non-empty subset of leaves of a complete binary tree $T$ with $n = 2^k$ leaves. The algorithm to calculate $M(\mathcal{R})$ in Table 6.1 terminates with output such that:

$$t(\mathcal{N}, M(\mathcal{R})) \geq t(\mathcal{N}, \mathcal{R})$$

*Proof.* The algorithm to calculate $M(\mathcal{R})$ terminates when the set $S$ is empty. $S$ is the set of triples $(v_i, v_j, v_k)$ where $v_i, \ldots, v_j$ is a Special Path and $v_k$ is a degree 3 node (or leaf) that is a child of $v_j$ (so $v_k$ is a null Special Path descended from the end node of a Special Path). As shown in Lemma 81, each pass through the central while loop does one of two things. $ST(\mathcal{R}')$ at the end

of the pass will differ from $ST(\mathcal{R}')$ at the start of the pass in that it will have either more Special Paths, or one null Special Path distance 2 closer to the root. Since the sum of the number of Special Paths and null Special Paths is $2r-1$ (Lemma 73) and the number of leaves in $\mathcal{R}'$ does not change in any pass through the loop, more Special Paths in the subtree means less null Special Paths in the subtree. So there can only be at most $2r-1$ passes through the loop that increase the number of Special Paths, since after that there will be no nodes $v_k$ that are null Special Paths and so $S$ will be empty.

The number of times the while loop does not increase the number of Special Paths is also bounded. Once a null Special Path, $v$, is high enough in $ST(\mathcal{R}')$ that there are no Special Paths on the path from $v$ to the root, $v$ cannot be a node in the form of $v_k$ for any triple in $S$. If the number of Special Paths stays the same in a pass through the while loop, then some null Special Path is replaced in $ST(\mathcal{R}')$ with one distance 2 closer to the root. After at most $\lfloor \frac{k}{2} \rfloor$ iterations applied to one null Special Path, it would be high enough in $ST(\mathcal{R})$ that it could not be in the form of $v_k$. The algorithm does not pick triples from $S$ in a way that "moves up" a given null Special Path, but selects them at random. However, since the number of null Special Paths in $ST(\mathcal{R}')$ is bounded by $2r-1$, the number of passes through the central while loop that do not increase the number of Special Paths is bounded by $(2r-1)\lfloor \frac{k}{2} \rfloor$. So the total number of passes through the while loop before $S$ is empty is bounded above by $2r-1+(2r-1)\lfloor \frac{k}{2} \rfloor = (2r-1)(\lfloor \frac{k}{2} \rfloor + 1)$. Therefore, the algorithm to calculate $M(\mathcal{R})$ terminates for any input $\mathcal{R}$. The number of Special Paths does not decrease in any one pass through the loop (Lemma 81), so it will not decrease for several passes. Therefore:

$$t(\mathcal{N}, M(\mathcal{R})) \geq t(\mathcal{N}, \mathcal{R}) \qquad \square$$

We can now describe the properties common to all $M$-type subsets. If there are any null Special Paths in $ST(M(\mathcal{R}))$, then they will occur at the top of the subtree, i.e. on the path from any one null Special Path to the root, there are only other null Special Paths and no Special Paths. If there were any Special Paths between a null Special Path and the root, then there would be some null Special Path descended from the end node of a Special Path

somewhere between the two. This cannot be the case for any output of the function $M$ as the algorithm only terminates when there are no such nodes. So along any path from the root to any leaf in $M(\mathcal{R})$, if the root has degree 2 we must have nodes of degree 3 all the way until the first Special Path. If the root has degree 1 (i.e. not a null Special Path) then there are no null Special Paths in $ST(M(\mathcal{R}))$. Plus, all Special Paths below the null Special Paths are separated by exactly one node of degree 3.

The function $M$ can be applied to all non-empty subsets $\mathcal{R} \subseteq \mathcal{N}$. If we apply it to any subset with $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ then we will also have $t(\mathcal{N}, M(\mathcal{R})) = t_{\max}(n, r)$. We cannot have $t(\mathcal{N}, M(\mathcal{R})) > t_{\max}(n, r)$ since $t_{\max}(n, r)$ is the maximum, nor can we have $t(\mathcal{N}, M(\mathcal{R})) < t_{\max}(n, r)$ by Corollary 82. So there exists an $M$-type subset that has maximum $t(\mathcal{N}, \mathcal{R})$, i.e. a subset where all Special Paths in $ST(\mathcal{R})$ are separated by exactly one node of degree 3.

As mentioned earlier, the size of the cover of an $M$-type subset can be counted more easily than with general subsets. This is shown in the following Lemma, which is a combination of preceding results.

*Lemma* 83. Let $SDRS$ be a Subset Difference Revocation Scheme with $n = 2^k$ users, defined on the complete binary tree $T$. Let $\mathcal{R}$ be any subset of $\mathcal{N}$. Then $t(\mathcal{N}, M(\mathcal{R})) = 2r - j$, where $j$ is the number of Special Subtrees in $ST(M(\mathcal{R}))$.

*Proof.* By the definition of the function $M$, all paths from the root to any leaf in $M(\mathcal{R})$ have null Special Paths first, and then Special Paths. For some leaf $v_l$, let $v$ be the first node in the first Special Path on the path from the root to $v_l$. The node $v$ is also the first node in a Special Path for all descendants of $v$ in $M(\mathcal{R})$, not just $v_l$. All descendants of $v$ in $ST(M(\mathcal{R}))$ have the Special Path containing $v$ on the path from their leaf to the root because $v$ only has one child in $ST(M(\mathcal{R}))$. If there were any Special Path above $v$, then that Special Path would also be on the path from the root to $v_l$, contradicting the definition of $v$. By the nature of the Steiner tree of an $M$-type subset, and because $v$ is in a Special Path, there are only Special Paths descended from $v$ along any path. This makes the portion of $ST(M(\mathcal{R}))$ descended from $v$ a Special Subtree. By Lemma 73 and the definition of a Special Subtree, there are $2r_1 - 1$ Special Paths in $ST(M(\mathcal{R}))$ descended from $v$, where $r_1$ is the

number of leaves in $M(\mathcal{R})$ descended from $v$. Let $set_1 = \{v_{i_1}, v_{i_2}, \ldots, v_{i_j}\}$ be the set of all nodes in $ST(M(\mathcal{R}))$ with the same properties as $v$, i.e. a node in a Special Path in $ST(M(\mathcal{R}))$ whose parent is a null Special Path. Since the number of leaves in $ST(M(\mathcal{R}))$ is just $|\mathcal{R}|$, the number of Special Paths descended from all the nodes in $set_1$ is just $2|\mathcal{R}| - |set_1| = 2|\mathcal{R}| - j$. Because all Special Paths in $ST(M(\mathcal{R}))$ must occur below some null Special Path in the subtree, every Special Path in $ST(M(\mathcal{R}))$ must be descended from some node in $set_1$. Therefore:

$$t(\mathcal{N}, \mathcal{R}) = 2r - j \qquad \square$$

So the size of the cover of $\mathcal{N} \setminus \mathcal{R}$ in a Subset Difference Revocation Scheme, where $\mathcal{R}$ is an $M$-type subset, is just $2|\mathcal{R}|$ minus the number of Special Subtrees in $ST(\mathcal{R})$. Since there is at least one $M$-type subset, $M(\mathcal{R})$, that will have $t(\mathcal{N}, M(\mathcal{R})) = t_{\max}(n, r)$, in order to calculate $t_{\max}(n, r)$ we can just determine $M(\mathcal{R})$ such that $ST(M(\mathcal{R}))$ has the minimum number of Special Subtrees. To do this, we will need the following notation:

*Definition* 84. Let $\mathcal{R}$ be a subset on leaves of a complete binary tree $T$. We define three functions as follows:

- $L(h)$ is the number of leaves in $T$ descended from a node at height $h$
- $C(h)$ is the number of leaves descended from a node at height $h$ such that the Steiner Tree of those leaves, with this node as root, forms a Maximal Special Subtree

Finally, for $k, r, j$ positive integers with $j \leq r \leq 2^k$, let $\mathcal{H}_{k,r,j}$ be the set of ordered $j$-tuples $(h_1, \ldots, h_j)$ with the property that $1 \leq h_i \leq k$ for $i = 1, \ldots, j$ and

$$\sum_{i=1}^{j} L(h_i) = 2^k \text{ and } \sum_{i=1}^{j} C(h_i) \geq r. \tag{6.1}$$

Let $j_{k,r}$ be the minimum value of $j$ such that $\mathcal{H}_{k,r,j} \neq \emptyset$.

There is a correspondence between $j$-tuples in $\mathcal{H}_{k,r,j}$ and $M$-type subsets. By showing how to create the latter from the former, and using Lemma 83, we get another expression for $t(\mathcal{N}, \mathcal{R})$.

*Lemma* 85. Let $SDRS$ be a Subset Difference Revocation Scheme on $n = 2^k$ users, defined on the complete binary tree $T$. Let $(h_1, \ldots, h_j) \in \mathcal{H}_{k,r,j}$. Then there exists an $M$-type subset, $\mathcal{R}$, with $|\mathcal{R}| = r$ and $t(\mathcal{N}, \mathcal{R}) = 2r - j$.

*Proof.* Let $(h_1, \ldots, h_j) \in \mathcal{H}_{k,r,j}$. Define the set of nodes of $T$, $set_1$, as follows: for $i$ from 1 to $j$, add to $set_1$ the left most node at height $h_i$ that does not have any descendants in common with the descendants of any nodes already in $set_1$. The fact that $\sum_{i=1}^{j} 2^{h_i} = 2^k$ (since $(h_1, \ldots, h_j) \in \mathcal{H}_{k,r,j}$) means that the sum of the number of leaves descended from $j$ nodes at heights $h_1, \ldots, h_j$ equals the number of leaves in $T$. So we will only be able to pick the $j$ nodes as described if there were no "gaps", i.e. leaves in $T$ that are not descendant from any of the nodes in $set_1$. We will use induction to show that there are no gaps to the left of any of the nodes added to $set_1$.

Consider the first node added. Since $set_1$ is empty, we just add the left most node at height $h_1$. The left most child of $T$ is a descendant of this node, so there is no gap to the left of this node (no leaf in $T$ not a descendant of the node). Assume this is true for the first $x$ nodes added to $set_1$, i.e. we have added $x$ nodes to $set_1$, and there is no gaps to the left of any of the sets of descendants. We know that $\sum_{i=1}^{j} 2^{h_i} = 2^k$ and because the heights are ordered we know $h_i \geq h_{x+1}$ for $x + 1 \leq i \leq j$, which implies $2^{h_{x+1}}$ divides $2^{h_i}$ for $i$ in the same range. Therefore, we have both:

$$\sum_{i=1}^{j} 2^{h_i} = 2^k \equiv 0 \mod 2^{h_{x+1}} \quad \text{and} \quad \sum_{i=x+1}^{j} 2^{h_i} \equiv 0 \mod 2^{h_{x+1}}$$

$$\text{Hence} \quad \sum_{i=1}^{x} 2^{h_i} \equiv 0 \mod 2^{h_{x+1}}.$$

This last summation is just the number of leaves descended from the $x$ nodes in $set_1$. Because there are no gaps between these descendants, all these leaves are all the descendants of a certain number of nodes at height $h_{x+1}$. Therefore, the next node added to $set_1$ will be the next node in this level, and there will be no gap between these leaves and the descendants of the first $x$ nodes in $set_1$. By induction, there will be no gap to the left of any of the descendants of the nodes in $set_1$ when we have added all $j$ nodes. And because the sum of the number of leaves descended from the $j$ nodes equals the number of leaves

in $T$, there can be no gap to the right of the final node either. Therefore all leaves in $T$ are descended from some node in $set_1$.

To construct $ST(\mathcal{R})$ from $set_1$, we just make each node in $set_1$ the root of a Special Subtree. The number of leaves we can have in $ST(\mathcal{R})$ descended from these nodes and still have Special Subtrees is bounded by $\sum_{i=1}^{j} C(h_i)$. But because $(h_1, \ldots, h_j) \in \mathcal{H}_{k,r,j}$, we have that $r \leq \sum_{i=1}^{j} C(h_i)$, and so it is possible to add $r$ such leaves. If $r$ is strictly less than the sum of the $C(h_i)$'s, then it does not make a difference which node(s) in $set_1$ has less than $C(h_i)$ leaves in $ST(\mathcal{R})$ so long as there is at least one leaf descended from each node in $set_1$ (possible since $j \leq r$). So all $j$ nodes in $set_1$ are in $ST(\mathcal{R})$, and because there is no leaf in $T$ that is not descended from one of these nodes, all ancestors of the nodes in $set_1$ are null Special Paths (they must have both children in $ST(\mathcal{R})$). Because there are only Special Paths descended from the nodes in $set_1$, $ST(\mathcal{R})$ is an $M$-type subset. So by Lemma 83, $t(\mathcal{N}, \mathcal{R}) = 2r - j$. □

As a consequence of Lemma 85, for a $j$-tuple in $\mathcal{H}_{k,r,j}$ with $j = j_{k,r}$, the minimum value of $j$ such that $\mathcal{H}_{k,r,j}$ is non-empty, $2r - j$ is the maximum possible value of $t(\mathcal{N}, \mathcal{R})$. To prove this, we need to show that it is possible to create an $M$-type subset out of a $j$-tuple.

*Corollary* 86. Let $SDRS$ be a Subset Difference Revocation Scheme on $n = 2^k$ users, defined on the complete binary tree $T$. For all $1 \leq r \leq n/2$:

$$t_{\max}(n, r) = 2r - j_{k,r}$$

*Proof.* By the definition of $j_{k,r}$, there is some tuple $(h_1, \ldots, h_{j_{k,r}}) \in H_{k,r,j_{k,r}}$ that satisfies Formula (6.1). From Lemma 85, there exists some $M$-type subset $\mathcal{R}$ with $|\mathcal{R}| = r$ and $t(\mathcal{N}, \mathcal{R}) = 2r - j_{k,r}$. Suppose there was a subset $\mathcal{R}'$ with $|\mathcal{R}'| = r$ and $t(\mathcal{N}, \mathcal{R}') > t(\mathcal{N}, \mathcal{R})$. If we calculate the subset $M(\mathcal{R}')$, then from Lemma 83 we have a description of $ST(M(\mathcal{R}'))$, namely all Special Paths are in Special Subtrees rooted at a number of nodes, say $j$, in $T$. Let $(h'_1, \ldots, h'_j)$ be the heights of these nodes. All nodes above these are null Special Paths. Because all leaves in $M(\mathcal{R}')$ are in one of these Special Subtrees, we have $r \leq \sum_{i=1}^{j} C(h'_i)$. Since all leaves in $T$ are descended from one of the nodes we have $\sum_{i=1}^{j} L(h'_i) = 2^k$. Therefore, if we order $(h'_1, \ldots, h'_j)$ then we get a

$j$-tuple in $\mathcal{H}_{k,r,j}$. From Lemma 83, $t(\mathcal{N}, M(\mathcal{R}')) = 2r - j$. But we know

$$t(\mathcal{N}, M(\mathcal{R}')) \geq t(\mathcal{N}, \mathcal{R}') > t(\mathcal{N}, \mathcal{R}) = 2r - j_{k,r}.$$

Therefore $j < j_{k,r}$. This contradicts the fact that $j_{k,r}$ is the minimum such that $\mathcal{H}_{k,r,j}$ is non-empty. Therefore $t_{\max}(n, r) = 2r - j_{k,r}$. □

So the problem of finding $t_{\max}(n, r)$ can now be re-stated as the problem of finding $j_{k,r}$. If we find the minimum tuple that satisfies Formula (6.1), then we will have found $t_{\max}(n, r)$. Dealing with the $j$-tuples is much more manageable than trying to maximise $t(\mathcal{N}, \mathcal{R})$.

The formulae for $L(h)$ and $C(h)$ come straight from their definition, and Lemma 76:

$$L(h) = 2^h \quad \text{and} \quad C(h) = 2^{\lfloor \frac{h-1}{2} \rfloor}.$$

We have some useful properties of the two functions that stem directly from their formulae:

$$L(h+1) = 2L(h), \qquad L(h+2) = 4L(h),$$
$$C(h+1) = \begin{cases} 2C(h) & \text{if } h \text{ even} \\ C(h) & \text{if } h \text{ odd} \end{cases}, \qquad C(h+2) = 2C(h).$$

These give rise to some constraints on the minimum $j$.

*Lemma* 87. Let $j_{k,r}$ be the minimum value of $j$ such that $\mathcal{H}_{k,r,j} \neq \emptyset$. Let $H = (h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$. Any even integer occurs at most once in $H$. Also, if $j_{k,r} > 1$, then $h_1$ is odd.

*Proof.* Suppose there is some even integer $h_{i_1}$ that occurs twice in $H$. Define $H'$ to be the sorted tuple of the other $j_{k,r} - 2$ integers and $h_{i_1} + 1$ ($H'$ is a $j_{k,r} - 1$-tuple). Since $h_{i_1}$ is even we have:

$$2L(h_{i_1}) = L(h_{i_1} + 1) \qquad \text{and} \qquad 2C(h_{i_1}) = C(h_{i_1} + 1).$$
$$\text{So} \quad \sum_{h \in H'} L(h) = \sum_{h \in H} L(h) = 2^k$$
$$\text{and} \quad \sum_{h \in H'} C(h) = \sum_{h \in H} C(h) \geq r.$$

Therefore $H' \in \mathcal{H}_{k,r,j_{k,r}-1}$. But this contradicts the fact that $j_{k,r}$ is the minimum $j$ such that $\mathcal{H}_{k,r,j}$ is non-empty. Therefore any even integer in $H$ must occur only once.

Suppose $j_{k,r} > 1$ and $h_1$ is even. Because $h_1$ is the smallest integer in $H$, if $h_1$ occurs an odd number of times in $H$ we have:

$$\sum_{h \in H} L(h) \equiv 2^{h_1} \mod 2^{h_1+1}.$$

We know $\sum_{h\in H} L(h) = 2^k$, since $H \in \mathcal{H}_{k,r,j_{k,r}}$. Because $j_{k,r} > 1$, there is at least two terms in the sum, and since $h_1$ is the smallest, $2^{h_1} < 2^k$ which implies $h_1 + 1 \leq k$. So $\sum_{h\in H} L(h) \equiv 0 \mod 2^{h_1+1}$. Therefore, $h_1$ must occur an even number of times in $H$. But we have just shown that any even integer can only occur once. Therefore, the smallest integer, $h_1$, is odd. □

This describes some of the properties of a $j$-tuple in $\mathcal{H}_{k,r,j}$ when $j$ is minimum. In order to further pin down the value of $j$, we need the following, more general object:

*Definition* 88. For $k, r, j$ positive integers with $j \leq r \leq 2^k$, let $\mathcal{H}'_{k,r,j}$ be the set of ordered $j$-tuples $(h_1, \ldots, h_j)$ with the property that $1 \leq h_i \leq k$ for $i = 1, \ldots, j$ and

$$\sum_{i=1}^{j} L(h_i) \leq 2^k \text{ and } \sum_{i=1}^{j} C(h_i) \geq r. \tag{6.2}$$

Let $j'_{k,r}$ be the minimum value of $j$ such that $\mathcal{H}'_{k,r,j} \neq \emptyset$.

Since the only difference between $\mathcal{H}'_{k,r,j}$ and $\mathcal{H}_{k,r,j}$ is a more relaxed constraint on the sum of the $L(h_i)$'s in $\mathcal{H}'_{k,r,j}$, we have that $\mathcal{H}_{k,r,j} \subseteq \mathcal{H}'_{k,r,j}$. Any $j$-tuple that satisfies Formula (6.1) also satisfies Formula (6.2). There is no direct correlation between tuples in $\mathcal{H}'_{k,r,j}$ and $M$-type subsets (like there is for tuples in $\mathcal{H}_{k,r,j}$). However, we can prove that if $\mathcal{H}'_{k,r,j}$ is non-empty, then so is $\mathcal{H}_{k,r,j}$.

*Lemma* 89. Let $H' = (h'_1, \ldots, h'_j) \in \mathcal{H}'_{k,r,j}$. Then there exists some $j$-tuple $H \in \mathcal{H}_{k,r,j}$.

*Proof.* If $\sum_{h\in H'} L(h) = 2^k$, then $H' = H \in \mathcal{H}_{k,r,j}$. Otherwise, we write the difference $2^k - \sum_{h\in H'} L(h)$ in binary, that is as the sum of distinct powers of

two. Suppose $2^i$ is the smallest power of two in this expansion. This gives us:

$$\underbrace{2^i + \text{ higher powers}}_{2^k - \sum_{h \in H'} L(h)} + \underbrace{2^{h'_1} + 2^{h'_2} + \ldots + 2^{h'_j}}_{\sum_{h \in H'} L(h)} = 2^k.$$

We know that $2^i$ occurs only once in the binary expansion of $2^k - \sum_{h \in H'} L(h)$. In order for the left hand side of the above equation to add up to $2^k$, there must be a collection of terms in the sum of $L(h)$ that add up to $2^i$:

$$2^i = 2^{h'_1} + \ldots + 2^{h'_a} \quad \text{for some} \quad 1 \leq a \leq j.$$

Define $H = (h_1, \ldots, h_j)$ as follows:

$$h_i = \begin{cases} h'_i + 1 & \text{if } 1 \leq i \leq a \\ h'_i & \text{if } a < i \leq j \end{cases}.$$

The difference between $L(h_i)$ and $L(h'_i)$ is just $L(h_i) = 2^{h_i} = 2^{h'_i+1} = 2.2^{h'_i} = 2L(h'_i)$ if $1 \leq i \leq a$. Otherwise $L(h_i) = L(h'_i)$. Since $\sum_{i=1}^{a} L(h'_i) = 2^i$, we have that:

$$\begin{aligned} \sum_{i=1}^{a} L(h_i) = 2.\sum_{i=1}^{a} L(h'_i) &= 2^i + \sum_{i=1}^{a} L(h'_i) \\ \text{Hence} \quad \sum_{i=1}^{j} L(h_i) &= 2^i + \sum_{i=1}^{j} L(h'_i). \end{aligned}$$

So not only do we have that $\sum_{h \in H} L(h) \leq 2^k$, but we also know that the binary expansion of $2^k - \sum_{h \in H} L(h)$ has one less term than that of $2^k - \sum_{h \in H'} L(h)$. Since $C(h)$ is non-decreasing in $h$,

$$\sum_{i=1}^{j} C(h_i) \geq \sum_{i=1}^{j} C(h'_i) \geq r.$$

So $H \in \mathcal{H}'_{k,r,j}$ (from the definition, $H$ is obviously a $j$-tuple). By repeating the same process, we will eventually reach the stage where $2^k - \sum_{h \in H} L(h) = 0$, in which case $H \in \mathcal{H}_{k,r,j}$. □

*Corollary* 90. Let $j'_{k,r}$ be the minimum value of $j$ such that $\mathcal{H}'_{k,r,j} \neq \emptyset$. Let $j_{k,r}$ be the minimum value of $j$ such that $\mathcal{H}_{k,r,j} \neq \emptyset$. Then:

$$j'_{k,r} = j_{k,r}$$

*Proof.* The only difference between $\mathcal{H}_{k,r,j}$ and $\mathcal{H}'_{k,r,j}$ is that the former requires that $\sum_{h\in H} L(h) = 2^k$, while the latter only requires that $\sum_{h\in H} L(h) \leq 2^k$. Therefore $\mathcal{H}_{k,r,j} \subseteq \mathcal{H}'_{k,r,j}$. So for the minimum $j$ such that $\mathcal{H}_{k,r,j}$ is non-empty, $\mathcal{H}'_{k,r,j}$ is also non-empty and so $j_{k,r} \geq j'_{k,r}$. But by Lemma 89, for the minimum $j$ such that $\mathcal{H}'_{k,r,j}$ is non-empty, $\mathcal{H}_{k,r,j}$ is also non-empty. Therefore, $j_{k,r} \leq j'_{k,r}$ and so $j_{k,r} = j'_{k,r}$. □

We could replace $j_{k,r}$ in the formula $t_{\max}(n,r) = 2r - j_{k,r}$ with $j'_{k,r}$ and work out $t_{\max}(n,r)$ by finding the minimum value of $j$ such that there exists a $j$-tuple that satisfies Formula (6.2). However, we are going to use $\mathcal{H}'_{k,r,j}$ and Corollary 90 to prove existence results for $\mathcal{H}_{k,r,j}$. The following results are proved using the same method. We assume there exists some $j$-tuple in $\mathcal{H}_{k,r,j}$ with a certain property. We then construct a $j-1$-tuple in $\mathcal{H}'_{k,r,j-1}$, proving that $j \neq j'_{k,r}$. And from Corollary 90 we also have that $j \neq j_{k,r}$. Therefore, no tuple in $\mathcal{H}_{k,r,j_{k,r}}$ can have that property.

*Lemma* 91. Let $j_{k,r}$ be the minimum value of $j$ such that $\mathcal{H}_{k,r,j} \neq \emptyset$. Let $H = (h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$. Then $h_{j_{k,r}} - h_1 \leq 4$.

*Proof.* Since all $H \in \mathcal{H}_{k,r,j_{k,r}}$ are ordered tuples, we have that $h_1 = \min(H)$ and $h_{j_{k,r}} = \max(H)$. If $j_{k,r} = 1$, then the result is obviously true ($h_{j_{k,r}} = h_1$). Otherwise, by Lemma 87 we have $h_1 = 2h - 1$ ($h_1$ must be odd). Suppose $h_{j_{k,r}} - h_1 \geq 5$, which means $h_{j_{k,r}} \geq 2h + 4$. Since:

$$\sum_{h\in H} L(h) = 2^k \equiv 0 \mod 2^{2h+1},$$

we must have either $(2h-1, 2h-1, 2h-1, 2h-1)$ or $(2h-1, 2h-1, 2h)$ as sub-tuples of $H$, so we can write $H$ in the form:

$$H = (2h-1, 2h-1, h^*, \ldots, h_{j_{k,r}}), \quad \text{where} \quad h^* \in \{2h-1, 2h\}.$$

By definition, $C(h^*) = 2^{h-1} = C(2h-1)$ for both possible values of $h^*$. Let $H'$ be the sorted $j_{k,r} - 1$ tuple consisting of all the integers in $H$, but with $h_{j_{k,r}} - 1$, $h_{j_{k,r}} - 2$ and $h_{j_{k,r}} - 2$ in place of $h_{j_{k,r}}$, $h^*$, $2h-1$ and $2h-1$. Because $h_{j_{k,r}} - 1 \geq 2h + 3$, we have:

$$\begin{aligned} C(h_{j_{k,r}} - 1) \geq C(2h+3) = 2^{\lfloor \frac{2h+2}{2} \rfloor} = 2^{h+1} = 4.2^{h-1} &= 4.C(2h-1) \\ &> C(h^*) + 2.C(2h-1). \end{aligned}$$

Since $2C(h_{j_{k,r}} - 2) = C(h_{j_{k,r}})$, it follows that:

$$\begin{aligned} C(h_{j_{k,r}} - 1) + 2.C(h_{j_{k,r}} - 2) &> C(h_{j_{k,r}}) + C(h^*) + 2.C(2h-1) \\ \text{Hence} \quad \sum_{h \in H'} C(h) &> \sum_{h \in H} C(h) \geq r. \end{aligned}$$

Now:

$$L(h_{j_{k,r}} - 1) + 2.L(h_{j_{k,r}} - 2) = 2^{h_{j_{k,r}}-1} + 2.2^{h_{j_{k,r}}-2} = 2^{h_{j_{k,r}}} = L(h_{j_{k,r}}).$$

Since both $L(2h-1)$ and $L(h^*)$ are non-zero:

$$\begin{aligned} L(h_{j_{k,r}} - 1) + 2.L(h_{j_{k,r}} - 2) &< L(h_{j_{k,r}}) + L(h^*) + 2.L(2h-1) \\ \text{Hence} \quad \sum_{h \in H'} L(h) &< \sum_{h \in H} L(h) = 2^k. \end{aligned}$$

Therefore, $H' \in \mathcal{H}'_{k,r,j_{k,r}-1}$, and so the minimum value of $j$ such that $\mathcal{H}'_{k,r,j}$ is non-empty is at most $j_{k,r} - 1$. But by Corollary 90, $\mathcal{H}_{k,r,j_{k,r}-1}$ must also be non-empty. This is a contradiction, as $j_{k,r}$ is defined to be the smallest integer $j$ such that $\mathcal{H}_{k,r,j}$ is non-empty. Therefore any $j_{k,r}$-tuple in $\mathcal{H}_{k,r,j_{k,r}}$ must have $h_{j_{k,r}} - h_1 \leq 4$. □

*Lemma* 92. Let $j_{k,r}$ be the minimum value of $j$ such that $\mathcal{H}_{k,r,j} \neq \emptyset$. Let $H = (h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$. Then $h_{j_{k,r}} - h_1 \neq 3$.

*Proof.* Suppose $h_{j_{k,r}} - h_1 = 3$. Since $h_1$ and $h_{j_{k,r}}$ are distinct, we cannot have $j_{k,r} = 1$. By Lemma 87, $h_1$ must be odd, $h_1 = 2h - 1$ and so $h_{j_{k,r}} = 2h + 2$. Since:

$$\sum_{h \in H} L(h) = 2^k \equiv 0 \mod 2^{2h},$$

$2h - 1$ must occur an even number of times in $H$. So we can write $H$ as:

$$H = (2h-1, 2h-1, \ldots, 2h+2).$$

Let $H'$ be the sorted $j_{k,r} - 1$ tuple consisting of all the integers in $H$, but with $2h+1$ and $2h+1$ in place of $2h-1$, $2h-1$ and $2h+2$. Since $2.L(2h+1) = L(2h+2)$ we have:

$$\begin{aligned} 2.L(2h+1) &< L(2h+2) + 2.L(2h-1) \\ \text{Hence} \quad \sum_{h \in H'} L(h) &< \sum_{h \in H} L(h) = 2^k. \end{aligned}$$

And from the properties of $C(h)$, we have that $C(2h+1) = C(2h+2)$ and $C(2h+1) = 2.C(2h-1)$, so:

$$\begin{aligned} 2.C(2h+1) &= C(2h+2) + 2.C(2h-1) \\ \text{Hence} \quad \sum_{h \in H'} C(h) &= \sum_{h \in H} C(h) \geq r. \end{aligned}$$

Therefore, $H' \in \mathcal{H}'_{k,r,j_{k,r}-1}$, and so the minimum value of $j$ such that $\mathcal{H}'_{k,r,j}$ is non-empty is at most $j_{k,r} - 1$. But by Corollary 90, $\mathcal{H}_{k,r,j_{k,r}-1}$ must also be non-empty. This is a contradiction, as $j_{k,r}$ is defined to be the smallest integer $j$ such that $\mathcal{H}_{k,r,j}$ is non-empty. Therefore, any $j_{k,r}$-tuple in $\mathcal{H}_{k,r,j_{k,r}}$ must have $h_{j_{k,r}} - h_1 \neq 3$. □

The consequence of Lemmas 91 and 92 are that for $(h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$ we must have $h_{j_{k,r}} - h_1 = 0, 1, 2$ or 4. What we want to show is that there always exists a $j_{k,r}$-tuple in $\mathcal{H}_{k,r,j_{k,r}}$ with $h_{j_{k,r}} - h_1 \leq 2$. Since this is not the same as showing there is no tuple with the property $h_{j_{k,r}} - h_1 = 4$[1], the proof will be different to the previous two. We need to show a tuple with $h_{j_{k,r}} - h_1 = 2$ can be constructed from one with $h_{j_{k,r}} - h_1 = 4$. If our constructed tuple was in $\mathcal{H}'_{k,r,j_{k,r}}$, then there would be no guarantee that the difference between $h_1$ and $h_{j_{k,r}}$ would remain the same when translated into a tuple in $\mathcal{H}_{k,r,j_{k,r}}$. Therefore, we need to construct the tuple so that it is in $\mathcal{H}_{k,r,j_{k,r}}$.

*Lemma* 93. Let $j_{k,r}$ be the minimum value of $j$ such that $\mathcal{H}_{k,r,j} \neq \emptyset$. Then there exists some $j_{k,r}$-tuple $H = (h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$ with $h_{j_{k,r}} - h_1 \leq 2$.

*Proof.* Let $H' = (h'_1, \ldots, h'_{j_{k,r}})$ be any $j_{k,r}$-tuple in $\mathcal{H}_{k,r,j_{k,r}}$. By Lemma 91 $h'_{j_{k,r}} - h'_1 \leq 4$, and by Lemma 92 $h'_{j_{k,r}} - h'_1 \neq 3$. So $h'_{j_{k,r}} - h'_1 = 0, 1, 2$ or 4. If $h'_{j_{k,r}} - h'_1 \leq 2$, then the result is already true. Assume $h'_{j_{k,r}} - h'_1 = 4$. By Lemma 87, and the fact that $j_{k,r} \geq 2$ ($h'_1$ and $h'_{j_{k,r}}$ are distinct), $h'_1$ must be odd: $h'_1 = 2h-1$, $h'_{j_{k,r}} = 2h+3$. We also must have $2h-1$ occurring an even number of times in order to have $\sum_{h \in H'} L(h) = 2^k$.

Suppose $2h-1$ occurs 4 or more times in $H'$:

$$H' = (2h-1, 2h-1, 2h-1, 2h-1, \ldots, 2h+3).$$

[1] There are such tuples, see Figure 6.5.

Let $H$ be the sorted $j_{k,r}-1$-tuple consisting of all the integers in $H'$, but with four $2h+1$ in place of $2h+3$ and four $2h-1$. Since $L(2h+3)=4.L(2h+1)$ we have:

$$\sum_{h\in H} L(h) < \sum_{h\in H'} L(h) = 2^k.$$

Also, $2.C(2h+1)=C(2h+3)$ and $2.C(2h+1)=4.C(2h-1)$ so:

$$\sum_{h\in H} C(h) = \sum_{h\in H'} C(h) \leq r.$$

So $H \in \mathcal{H}'_{k,r,j_{k,r}-1}$, and the minimum value of $j$ such that $\mathcal{H}_{k,r,j} \neq \emptyset$ is at most $j_{k,r}-1$. This contradicts Corollary 90. Therefore, $2h-1$ only occurs twice in $H'$. The only way for $\sum_{h\in H'} L(h) = 2^k = 0 \mod 2^{2h+1}$ is to have $2h$ in $H'$ (but only one occurrence by Lemma 87):

$$H' = (2h-1, 2h-1, 2h, \ldots, 2h+3).$$

Let $H$ be the sorted $j_{k,r}$-tuple consisting of all the integers in $H'$, but with three $2h+1$ and $2h+2$ in place of two $2h-1$, $2h$ and $2h+3$. Since:

$$\begin{aligned} 2.L(2h-1)+L(2h)+L(2h+3) = 2.2^{2h-1}+2^{2h}+2^{2h+3} &= 2^{2h+3}+2^{2h+1} \\ \text{and} \quad 3.L(2h+1)+L(2h+2) = 3.2^{2h+1}+2^{2h+2} &= 2^{2h+3}+2^{2h+1} \\ \text{we have} \quad \sum_{h\in H'} L(h) = \sum_{h\in H} L(h) &= 2^k. \end{aligned}$$

Also:

$$\begin{aligned} 2.C(2h-1)+C(2h)+C(2h+3) &= 2.2^{\lfloor \frac{2h-2}{2} \rfloor} + 2^{\lfloor \frac{2h-1}{2} \rfloor} + 2^{\lfloor \frac{2h+2}{2} \rfloor} \\ &= 3.2^{h-1} + 2^{h+1} = 3.2^h + 2^{h-1} \\ \text{and} \quad 3.C(2h+1)+C(2h+2) &= 3.2^{\lfloor \frac{2h}{2} \rfloor} + 2^{\lfloor \frac{2h+1}{2} \rfloor} \\ &= 3.2^h + 2^h = 4.2^h \\ \text{which implies} \quad \sum_{h\in H'} C(h) > \sum_{h\in H} C(h) &\geq r. \end{aligned}$$

Therefore, $H \in \mathcal{H}_{k,r,j_{k,r}}$. We already showed that $2h-1$ could only occur twice, and $2h$ once, in $H'$. We defined $H$ to be the integers in $H'$ without $(2h-1, 2h-1, 2h, 2h+3)$ and with $(2h+1, 2h+1, 2h+1, 2h+2)$. So if we use the notation $H = (h_1, \ldots, h_{j_{k,r}})$, then $h_1$ must be $2h+1$. Since $h_i \leq 2h+3$ for all $h_i \in H'$, $h_{j_{k,r}} \leq 2h+3$. So $h_{j_{k,r}} - h_1 \leq 2$. □

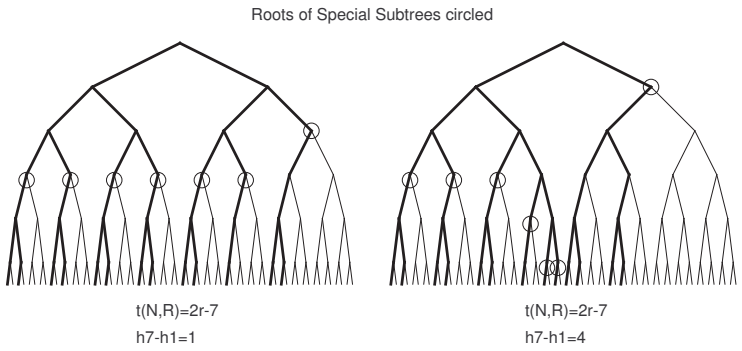


Figure 6.5: Two choices of $\mathcal{R}$ with $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$

Lemma 91 shows that for any $H \in \mathcal{H}_{k,r,j_{k,r}}$, $h_{j_{k,r}} - h_1 \leq 4$. If there is such a $j_{k,r}$-tuple with $h_{j_{k,r}} - h_1 = 4$, then the simple manipulation in Lemma 93 gives us a tuple where the greatest difference in the integers is less than or equal 2. How this relates back to $t_{\max}(n, r)$ is shown in Figure 6.5. In both diagrams, we have a complete binary tree of 64 users with 13 users revoked. $ST(\mathcal{R})$ is shown as thick lines. Both revoked sets have properties of an $M$-type subset (going from leaf to root we have a Special Subtree and then nodes with both children in $ST(\mathcal{R})$). On the left we have the roots of the Special Subtrees ($H = (3, 3, 3, 3, 3, 3, 4)$) are in two adjacent levels ($h_{j_{k,r}} - h_1 = 4 - 3 = 1$). On the right the difference in heights between the highest and lowest root is 4 ($H' = (1, 1, 2, 3, 3, 5)$, $h'_{j_{k,r}} - h'_1 = 5 - 1 = 4$). We can see from the diagram that in both cases $t(\mathcal{N}, \mathcal{R}) = 19 = 2(13) - 7$, which agrees with the formula $t(\mathcal{N}, \mathcal{R}) = 2r -$ (number of Special Subtrees). We will show later that both of these examples give $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$. What the diagram on the right shows is that we will not always get the range of the roots to be less than 4 for $t_{\max}(n, r)$. We are not trying to classify all subsets that give $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$, but we merely want to give sufficient conditions.

We know that there exists some tuple in $H \in \mathcal{H}_{k,r,j_{k,r}}$ with $h_{j_{k,r}} - h_1 \leq 2$. And from Lemma 87, either $j_{k,r} = 1$ or the smallest integer in $H$ is odd. Either way, we can write $H$ in the form:

$$H = (h_1, \ldots, h_{j_{k,r}}) = (\underbrace{2h+1, \ldots, 2h+1}_{\alpha}, \underbrace{2h+2}_{\epsilon}, \underbrace{2h+3, \ldots, 2h+3}_{\beta}), \quad (6.3)$$

where $\alpha, \beta \geq 0$ and $\epsilon \in \{0, 1\}$. The fact that $H \in \mathcal{H}_{k,r,j_{k,r}}$ means that the sum

of $\alpha + \epsilon + \beta = j_{k,r}$ and:

$$\alpha.2^{2h+1} + \epsilon.2^{2h+2} + \beta.2^{2h+3} = 2^k \quad (6.4)$$

$$\alpha.2^h + \epsilon.2^h + \beta.2^{h+1} \geq r. \quad (6.5)$$

Since there exists some tuple in $\mathcal{H}_{k,r,j_{k,r}}$ with a corresponding $\alpha, \beta$ and $\epsilon$, in order to find $j_{k,r}$, we just need to find the minimum $\alpha + \beta + \epsilon$ subject to the above constraints ($j_{k,r}$ is the minimum value such that there is some solution).

There is some ambiguity in this notation. If $H$ is a tuple of the same integer repeated (i.e. $h_1 = h_j$), then this can be represented by $\alpha$ occurrences of $2h+1$, or $\beta$ of $2h'+3$. In order to remove this ambiguity, we stipulate that $\alpha \neq 0$. We will use Formulae (6.4) and (6.5), to work out $\alpha, \beta$ and $\epsilon$. But first, we need to find the value of $h$.

*Lemma* 94. Let $H = (h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$ be any tuple with $h_{j_{k,r}} - h_1 \leq 2$. If $H$ is written in the form of Formula (6.3), then $h$ is the maximum value such that a feasible solution of Formulae (6.4) and (6.5) exists.

*Proof.* Let $h = h_1$ and $(\alpha, \beta, \epsilon) = (\alpha_1, \beta_1, \epsilon_1)$ be the solution to Formulae (6.4) and (6.5) that minimises $\alpha + \beta + \epsilon$. If we assume the statement of the Lemma is false, then there exists another solution $h = h_2$, $(\alpha, \beta, \epsilon) = (\alpha_2, \beta_2, \epsilon_2)$, with $h_2 \geq h_1 + 1$ and $\alpha_1 + \beta_1 + \epsilon_1 \leq \alpha_2 + \beta_2 + \epsilon_2$. By Formula (6.4):

$$\begin{aligned}
\alpha_1 2^{2h_1+1} + \epsilon_1 2^{2h_1+2} + \beta_1 2^{2h_1+3} &= 2^k \\
\alpha_1 + 2\epsilon_1 + 4\beta_1 &= 2^{k-2h_1-1}. \quad (6.6) \\
\text{Similarly} \quad \alpha_2 + 2\epsilon_2 + 4\beta_2 &= 2^{k-2h_2-1} \\
h_2 \geq h_1 + 1 \quad \text{so} \quad \alpha_2 + 2\epsilon_2 + 4\beta_2 &\leq 2^{k-2(h_1+1)-1} \\
&= 2^{k-2h_1-3} = \frac{2^{k-2h_1-1}}{4} \\
\text{Substituting back in (6.6) gives} \quad \alpha_2 + 2\epsilon_2 + 4\beta_2 &\leq \frac{\alpha_1 + 2\epsilon_1 + 4\beta_1}{4} \\
\text{i.e.} \quad 4\alpha_2 + 8\epsilon_2 + 16\beta_2 &\leq \alpha_1 + 2\epsilon_1 + 4\beta_1.
\end{aligned}$$

We know $\alpha_1 > 0$, so adding $3\alpha_1 + 2\epsilon_1$ to the right hand side of this inequality

gives:

$$\begin{aligned} 4\alpha_2 + 8\epsilon_2 + 16\beta_2 &< 4\alpha_1 + 4\epsilon_1 + 4\beta_1 \\ \text{Hence} \quad 4\alpha_2 + 4\epsilon_2 + 4\beta_2 &< 4\alpha_1 + 4\epsilon_1 + 4\beta_1 \\ \text{and} \quad \alpha_2 + \epsilon_2 + \beta_2 &< \alpha_1 + \epsilon_1 + \beta_1. \end{aligned}$$

This contradicts the assumption that $\alpha_1 + \epsilon_1 + \beta_1$ is minimised. Therefore, the triple $(\alpha, \epsilon, \beta)$ that minimises $\alpha + \epsilon + \beta$ subject to Formulae (6.4) and (6.5) must solve them for the maximum $h$ such that a feasible solution exists. □

*Corollary* 95. Let $H = (h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$ be any tuple with $h_{j_{k,r}} - h_1 \leq 2$. If $H$ is written in the form of Formula (6.3), then $h = k - u - 2$ where $2^u < r \leq 2^{u+1}$.

*Proof.* If we multiply Formula (6.5) by $2^{h+1}$, we get:

$$\begin{aligned} & \alpha 2^h + \epsilon 2^h + \beta 2^{h+1} \geq r \\ & \alpha 2^{2h+1} + \epsilon 2^{2h+1} + \beta 2^{2h+2} \geq r.2^{h+1} \\ \text{Formula (6.4)} \quad & \alpha 2^{2h+1} + \epsilon 2^{2h+2} + \beta 2^{2h+3} = 2^k \\ \text{Subtract the two} \quad & \epsilon 2^{2h+1} + \beta 2^{2h+2} \leq 2^k - r.2^{h+1} \\ \text{So that} \quad & \epsilon + 2\beta \leq 2^{k-2h-1} - r.2^{-h} \\ \text{i.e.} \quad & \epsilon + 2\beta \leq \frac{2^{k-h-1} - r}{2^h}. \end{aligned}$$

Let $u$ be such that $2^u < r \leq 2^{u+1}$. Suppose $h \geq k - u - 1$. Then:

$$\frac{2^{k-h-1} - r}{2^h} \leq \frac{2^u - r}{2^{k-u-1}} < 0.$$

Since $\epsilon + 2\beta \leq \frac{2^{k-h-1}-r}{2^h}$, and $\epsilon$ and $\beta$ are both non-negative, the fact that $\frac{2^{k-h-1}-r}{2^h} < 0$ means there are no feasible solutions for $\epsilon$ and $\beta$.

Let $h = k - u - 2$ (so $k - h - 1 = u + 1$). This gives:

$$\frac{2^{k-h-2} - r}{2^h} = \frac{2^{u+1} - r}{2^{k-u-2}} \geq 0.$$

Since this fraction is greater than or equal to zero, we have at least on solution: $\epsilon = 0$ and $\beta = 0$. Therefore, $h = k - u - 2$ is the maximum value such that a feasible solution to Formulae (6.4) and (6.5) exists. By Lemma 94:

$$h = k - u - 2.$$ □

Now that we know $h$, we can place a condition on the values of $\alpha, \beta$ and $\epsilon$.

*Lemma* 96. Let $H = (h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$ be any tuple with $h_{j_{k,r}} - h_1 \leq 2$. If $H$ is written in the form of Formula (6.3), then $\epsilon + 2\beta \geq \epsilon_1 + 2\beta_1$, where $(\alpha_1, \beta_1, \epsilon_1)$ is any other solution to Formulae (6.4) and (6.5).

*Proof.* Suppose $(\alpha, \beta, \epsilon)$ is the minimum solution to Formulae (6.4) and (6.5). Let $(\alpha_1, \beta_1, \epsilon_1)$ be any other solution (so $\alpha_1 + \beta_1 + \epsilon_1 \geq \alpha + \beta + \epsilon$) but with $\epsilon_1 + 2\beta_1 > \epsilon + 2\beta$. Therefore:

$$\begin{aligned} (\epsilon + 2\beta) + \gamma &= (\epsilon_1 + 2\beta_1), \quad \text{for some } \gamma > 0 \\ \text{So} \quad 2\beta - 2\beta_1 &= -\gamma + (\epsilon_1 - \epsilon) \\ \text{or} \quad \beta - \beta_1 &= \frac{-\gamma + (\epsilon_1 - \epsilon)}{2}. \end{aligned} \tag{6.7}$$

Because $(\alpha, \epsilon, \beta)$ and $(\alpha_1, \epsilon_1, \beta_1)$ both solve Formula (6.4), we have:

$$\begin{aligned} \alpha.2^{2h+1} + \epsilon.2^{2h+2} + \beta.2^{2h+3} &= 2^k \\ \alpha_1.2^{2h+1} + \epsilon_1.2^{2h+2} + \beta_1.2^{2h+3} &= 2^k \\ \text{Hence} \quad \alpha.2^{2h+1} + \epsilon.2^{2h+2} + \beta.2^{2h+3} &= \alpha_1.2^{2h+1} + \epsilon_1.2^{2h+2} + \beta_1.2^{2h+3} \\ \text{Thus} \quad \alpha + 2\epsilon + 4\beta &= \alpha_1 + 2\epsilon_1 + 4\beta_1 \\ \text{i.e.} \quad \alpha + 2(\epsilon + 2\beta) &= \alpha_1 + 2(\epsilon_1 + 2\beta_1) \\ \text{So} \quad \alpha + 2(\epsilon + 2\beta) &= \alpha_1 + 2(\epsilon + 2\beta + \gamma) \end{aligned}$$

Subtracting $2(\epsilon + 2\beta)$ gives

$$\begin{aligned} \alpha &= \alpha_1 + 2\gamma \\ \text{or} \quad \alpha - \alpha_1 &= 2\gamma \end{aligned} \tag{6.8}$$

Combining Equation (6.7) and Equation (6.8) gives:

$$\alpha + \beta - (\alpha_1 + \beta_1) = 2\gamma + \frac{-\gamma + (\epsilon_1 - \epsilon)}{2}.$$

If $\epsilon = \epsilon_1$, then:

$$\begin{aligned} \alpha + \beta - (\alpha_1 + \beta_1) &= \frac{3\gamma}{2} > 0 \\ \text{So} \quad \alpha + \beta + \epsilon - (\alpha_1 + \beta_1 + \epsilon_1) &> 0 \\ \text{i.e.} \quad \alpha + \beta + \epsilon &> \alpha_1 + \beta_1 + \epsilon_1. \end{aligned}$$

Otherwise, $\epsilon_1 = 1 - \epsilon$, either because $\epsilon = 1$ and $\epsilon_1 = 0$ or $\epsilon = 0$ and $\epsilon_1 = 1$. This gives:

$$\begin{aligned}
\alpha + \beta - (\alpha_1 + \beta_1) &= 2\gamma + \frac{-\gamma + (1 - \epsilon - \epsilon)}{2} \\
&= \frac{3\gamma}{2} + \frac{1}{2} - \epsilon_1 \\
\alpha + \beta + \epsilon - (\alpha_1 + \beta_1) &= \frac{3\gamma + 1}{2} > 1 && \text{since } \gamma > 0 \\
\alpha + \beta + \epsilon - (\alpha_1 + \beta_1) &> 1 \\
\alpha + \beta + \epsilon - (\alpha_1 + \beta_1 + \epsilon_1) &> 0 && \text{since } \epsilon \leq 1 \\
\alpha + \beta + \epsilon &> \alpha_1 + \beta_1 + \epsilon_1.
\end{aligned}$$

In both cases, we have $\alpha+\beta+\epsilon > \alpha_1+\beta_1+\epsilon_1$, which contradicts the assumption on both solutions. Therefore, if $\alpha, \beta, \epsilon$ is the solution to Formulae (6.4) and (6.5) that minimises $\alpha + \beta + \epsilon$, then $\alpha, \beta, \epsilon$ maximises $\epsilon + 2\beta$. □

Corollary 95 and Lemma 96 are enough to uniquely determine $\alpha, \beta$ and $\epsilon$.

*Theorem* 97. Let $H = (h_1, \ldots, h_{j_{k,r}}) \in \mathcal{H}_{k,r,j_{k,r}}$ be any tuple with $h_{j_{k,r}} - h_1 \leq 2$. If $H$ is written in the form of Formula (6.3), then:

$$\alpha + \epsilon + \beta = 2^{2u+3-k} - \delta - 3m,$$

where $2^u < r \leq 2^{u+1}$, and $2^{u+1} - r = 2^{k-u-1}.m + 2^{k-u-2}.\delta + r'$, with $\delta \in \{0, 1\}$ and $0 \leq r' < 2^{k-u-2}$.

*Proof.* By Lemma 96, $\alpha, \beta, \epsilon$ are solutions to Formulae (6.4) and (6.5) that maximise $\epsilon + 2\beta$. Since $\epsilon \in \{0, 1\}$, this means it is enough to maximise $\beta$ independent from $\epsilon$ ($\epsilon + 2\beta$ is maximum when $\beta$ is maximum). From Corollary 95, and the fact that $h = k - u - 2$, the constrains on $\alpha, \beta, \epsilon$ can be written as:

$$\begin{aligned}
\epsilon + 2\beta &\leq \frac{2^{k-h-1} - r}{2^h} \\
\epsilon + 2\beta &\leq \frac{2^{u+1} - r}{2^{k-u-2}} && \text{(since } k - h - 1 = u + 1\text{)} \\
2\beta &\leq \frac{2^{u+1} - r}{2^{k-u-2}} && \text{(since we are maximising } \beta\text{)}.
\end{aligned}$$

$\beta$ must be the greatest integer that satisfies this inequality. By re-writing $2^{u+1} - r$ in the form $2^{u+1} - r = 2^{k-u-1}.m + 2^{k-u-2}.\delta + r'$, with $\delta \in \{0, 1\}$ and

$0 \leq r' < 2^{k-u-2}$, we get:

$$\begin{aligned}
2\beta &\leq \frac{2^{u+1} - r}{2^{k-u-2}} \\
\beta &\leq \frac{2^{k-u-1}.m + 2^{k-u-2}.\delta + r'}{2^{k-u-1}} \\
\beta &= \left\lfloor \frac{2^{k-u-1}.m}{2^{k-u-1}} + \frac{2^{k-u-2}.\delta + r'}{2^{k-u-1}} \right\rfloor && \text{(since } \beta \in \mathbb{N}\text{)} \\
&= \left\lfloor m + \frac{\delta}{2} + \frac{r'}{2^{k-u-1}} \right\rfloor \\
&= m + \left\lfloor \frac{\delta}{2} + \frac{r'}{2^{k-u-1}} \right\rfloor && \text{(since } m \in \mathbb{N}\text{)} \\
&= m,
\end{aligned}$$

since $r' < 2^{k-u-2}$ which implies $\frac{r'}{2^{k-u-2}} < \frac{1}{2}$ and $\delta \in \{0, 1\}$ which implies $\frac{\delta}{2} \leq \frac{1}{2}$. With the value of $\beta$ fixed, the value of $\epsilon$ that maximises $\epsilon + 2\beta$ is:

$$\begin{aligned}
\epsilon &= \left\lfloor \frac{2^{u+1} - r}{2^{k-u-2}} \right\rfloor - 2\beta \\
&= \left\lfloor \frac{2^{k-u-1}.m + 2^{k-u-2}.\delta + r'}{2^{k-u-2}} \right\rfloor - 2m \\
&= \left\lfloor \frac{2^{k-u-1}.m}{2^{k-u-2}} + \frac{2^{k-u-2}.\delta}{2^{k-u-2}} + \frac{r'}{2^{k-u-2}} \right\rfloor - 2m \\
&= 2m + \delta + \left\lfloor \frac{r'}{2^{k-u-2}} \right\rfloor - 2m \\
&= \delta \quad \text{since } r' < 2^{k-u-2}.
\end{aligned}$$

Now that we have the values of $\epsilon$ and $\beta$, we can use Formula (6.4) to work out $\alpha$:

$$\begin{aligned}
\alpha.2^{2h+1} + \epsilon.2^{2h+2} + \beta.2^{2h+3} &= 2^k \\
\alpha + 2\epsilon + 4\beta &= 2^{k-2h-1} \\
\alpha + 2\epsilon + 4\beta &= 2^{k-2(k-u-2)-1} \\
\alpha + 2\epsilon + 4\beta &= 2^{2u-k+3} \\
\alpha &= 2^{2u-k+3} - 2\epsilon - 4\beta \\
\text{So} \quad \alpha + \epsilon + \beta &= 2^{2u-k+3} - \epsilon - 3\beta \\
&= 2^{2u-k+3} - \delta - 3m. \qquad \square
\end{aligned}$$

| Range of $r$ | $t_{\max}(n,r)$ | Proof |
|---|---|---|
| $r=0$ | 1 | - |
| $1 \leq r < 2^{\lfloor \frac{k-1}{2} \rfloor}$ | $2r-1$ | Lemma 78 |
| $2^{\lfloor \frac{k-1}{2} \rfloor} \leq r < 2^{\lceil \frac{k-1}{2} \rceil}$ | $2r-2$ | Corollary 79 |
| $2^{\lceil \frac{k-1}{2} \rceil} \leq r < 2^{k-1}$ | Formula (6.9) | Corollary 98 |
| $2^{k-1} \leq r < 2^k$ | $2^k - r$ | Corollary 6 |

Table 6.2: Complete formulae for $t_{\max}(n,r)$

*Corollary* 98. Let $SDRS$ be a Subset Difference Revocation Scheme with $n = 2^k$ users. Let $r$ be the number of revoked users, $2^{\lceil \frac{k-1}{2} \rceil} \leq r \leq 2^{k-1}$. Let $u$ be the integer such that $2^u < r \leq 2^{u+1}$. Then $SDRS$ has:

$$t_{\max}(n,r) = 2r - (2^{2u-k+3} - \delta - 3m), \tag{6.9}$$

where $2^{u+1} - r = 2^{k-u-1}.m + 2^{k-u-2}.\delta + r'$, with $\delta \in \{0,1\}$ and $0 \leq r' < 2^{k-u-2}$.

*Proof.* By Corollary 86 $t_{\max}(n,r) = 2r - j_{k,r}$. By Lemma 93, there exists some $j_{k,r}$-tuple $H \in \mathcal{H}_{k,r,j_{k,r}}$ with $h_{j_{k,r}} - h_1 \leq 2$, and so $H$ can be written in the form of Formula (6.3). In this form $j_{k,r} = \alpha + \epsilon + \beta$. The formula for $\alpha + \epsilon + \beta$ for such a $j_{k,r}$-tuple, as proved in Theorem 97, is:

$$\alpha + \epsilon + \beta = 2^{2u-k+3} - \delta - 3m$$

By Formula (6.3) $\quad t_{\max}(n,r) = 2r - (2^{2u-k+3} - \delta - 3m).$ □

If we return to the example in Figure 6.5, we can see how this formula is applied. In the example we had $n = 64 = 2^6$, and $r = 13$. Since $2^3 < r \leq 2^4$, we have $u = 3$. The value of $2^{u+1} - r = 3$, and we want to write this in the form of $2^{k-u-1}.m + 2^{k-u-2}.\delta + r'$. Since $2^{k-u-2} = 2$ and $2^{k-u-1} = 4$, $3 = 4.m + 2.\delta + r'$ when $m = 0$, $\delta = 1$ and $r' = 1$. This gives $j_{k,r} = 2^{2u-k+3} - \delta - 3m = 8 - 1 = 7$. Since both Steiner Trees in Figure 6.5 are made up of 7 Special Subtrees, this means $t_{\max}(n,r) = t(\mathcal{N},\mathcal{R})$, which in this case is $2(13) - 7 = 19$.

The various formulae for $t_{\max}(n,r)$ over the complete range of $r = [0,\ldots,n]$ are given in Table 6.2. Figure 6.6 is a graph of $t_{\max}(n,r)$ for a large population, $n = 1024$, along with the original bound of $2r-1$ ([23]). The figure gives an indication of how pessimistic the bound is. The bound and $t_{\max}(n,r)$ start off

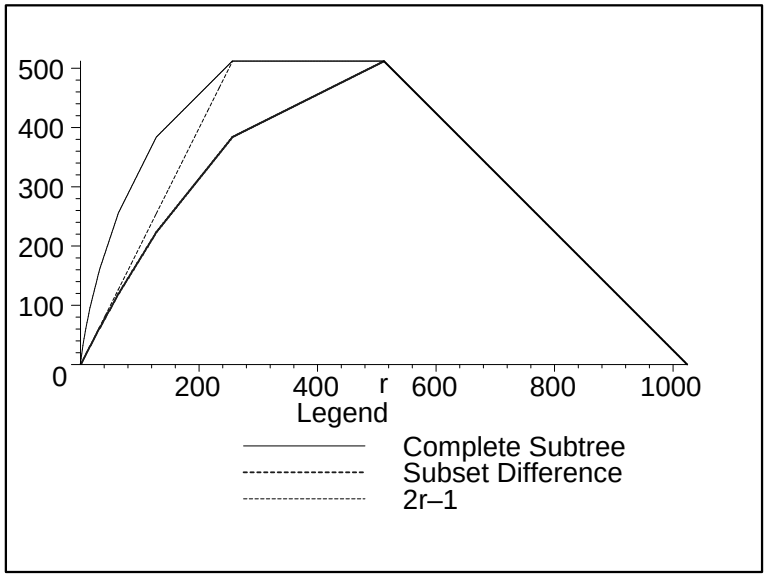


Figure 6.6: $t_{\max}(n,r)$, Complete Subtree and Subset Difference, $n = 1024$

as the same line, but we showed in Lemma 76 that this is only the case for $r \leq 2^{\lfloor \frac{k-1}{2} \rfloor} \approx \sqrt{n}$. After this, the two diverge, eventually being separated by a distance of almost $n/8$: $(2(n/4)-1)-t_{\max}(n,n/4) = (n/2-1)-3n/8 = n/8-1$. For $r > n/4$, $\min(n/2, n-r)$ is the more appropriate upper bound on $t_{\max}(n,r)$.

Also shown in Figure 6.6 is $t_{\max}(n,r)$ for the Complete Subtree Revocation Scheme. In [23], and other related papers, the maximum bandwidth cost of the two schemes have only been compared in terms of upper bounds. The respective formula we have derived now allow us to compare the actual maximum bandwidth of the Subset Difference Revocation Scheme ($SDRS$) and the Complete Subtree Revocation Scheme ($CSRS$). We see that for small values of $r$, $t_{\max}^{SDRS}(n,r)$ is significantly less than $t_{\max}^{CSRS}(n,r)$ ($2r-1$ versus $\approx r(\log_2(n/r))$). In the range of $n/4 \leq r \leq n/2$, $t_{\max}^{SDRS}(n,r)$ is a step function with $t_{\max}^{SDRS}(n,r)$ increasing from $3n/8$ to $n/2$, as compared to a constant $t_{\max}^{CSRS}(n,r) = n/2$. As stated earlier, both schemes have the same value of $t_{\max}(n,r)$ for all $r$ greater than $n/2$, $t_{\max}^{SDRS}(n,r) = t_{\max}^{CSRS}(n,r) = n-r$. This means that the Subset Difference Revocation Scheme has higher maximum bandwidth than the Forest of Trees Revocation Schemes in this range. The Forest of Trees Revocation Schemes achieved $t_{\max}(n,r)$ significantly lower than $n-r$, as shown in Formula (5.12). There will be a more thorough comparison

of all schemes presented in Chapter 7.

## 6.3 Average Bandwidth

In the Chapter 5, we found a formula for $t_{aver}(n, r)$ for the disjoint union of two revocation schemes (Formula (5.5)). As we mentioned in the previous section, the Subset Difference Revocation Scheme on $2n$ users is not just comprised of the disjoint union of two schemes on $n$ users. There is more than one subset in the Subset Difference Revocation Scheme on $2n$ users that contains users from both of the component Subset Difference Revocation Schemes on $n$ users. However, if we can specify all such subsets, we may be able to determine the difference in the sizes of the covers of the Subset Difference Revocation Scheme with $2n$ users ($SDRS$) and the disjoint union of two Subset Difference Revocation Schemes with $n$ users ($DURS$). Formula (5.5) gives us a formula for the $t_{aver}(2n, r)$ for the latter scheme:

$$t_{aver}^{DURS}(2n, r) = \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \frac{\binom{n}{r_1}\binom{n}{r-r_1}\left(t_{aver}^{SDRS}(n, r_1) + t_{aver}^{SRDS}(n, r - r_1)\right)}{\binom{2n}{r}}. \tag{6.10}$$

Note that this is not a recursive relation as it expresses $t_{aver}(2n, r)$ for one scheme ($DURS$) in terms of $t_{aver}(n, r)$ of another scheme ($SDRS$) with a smaller population. More importantly, we wish to find a recursive relation for $t_{aver}(n, r)$ with $SDRS$. However, if we can classify all the subsets $\mathcal{R}$ such that the size of the cover of $\mathcal{N} \setminus \mathcal{R}$ in $SDRS$ ($t^{SDRS}(\mathcal{N}, \mathcal{R})$) differs from the cover in $DURS$ ($t^{DURS}(\mathcal{N}, \mathcal{R})$), and by how much they differ, then we can modify Formula (6.10) to give a recursive formula for $t_{\max}^{SDRS}(n, r)$. Because $SDRS$ contains all the subsets in $DURS$, we know that $t^{SDRS}(\mathcal{N}, \mathcal{R}) \leq t^{DURS}(\mathcal{N}, \mathcal{R})$.

Let $SDRS = (\mathcal{N}, \Omega, \gamma)$ be a Subset Difference Revocation Scheme on a set of $2n$ users determined by the correspondence between the users and the leaves of a complete binary tree $T$. Let $\mathcal{N}_1$ be the set of users whose leaves are descended from the left child of the root of $T$, and $\mathcal{N}_2$ be the set of users whose leaves are descended from the right child of the root of $T$. Let $SDRS_1 = (\mathcal{N}_1, \Omega_1, \gamma_1)$ and $SDRS_2 = (\mathcal{N}_2, \Omega_2, \gamma_2)$ be Subset Difference Revocation Schemes defined on the two subtrees of $T$, $T_1$ and $T_2$, rooted at

the left and right child of the root of $T$. In the previous section we showed that all the subsets in $SDRS_1$ and $SDRS_2$ (i.e. $\{\gamma_1(i,j) : (i,j) \in \Omega_1\}$ and $\{\gamma_2(i,j) : (i,j) \in \Omega_2\}$), occur in $SDRS$. We need to classify the subsets, $\gamma(i,j)$, that occur in $SDRS$, but do not occur in either $SDRS_1$ or $SDRS_2$.

In the following Lemma, we will prove the following: for any $\gamma(i,j) \in SDRS$, we have $\gamma(i,j) \in SDRS_b$ if and only if $i$ is a node of $T_b$ or $i$ is the root of $T$ and $j$ is the root of $T_{1-b}$ (for $b \in \{0,1\}$). Conversely, if $\gamma(i,j) \in SDRS$, then $\gamma(i,j) \notin SDRS_1$ and $\gamma(i,j) \notin SDRS_2$ if and only if $(i,j) = (0,0)$ or $i$ is the root of $T$ and $j$ is not a child of the root of $T$.

*Lemma* 99. Let $T$ be a complete binary tree with $2n \geq 4$ leaves. Let $SDRS_1 = (\mathcal{N}_1, \Omega_1, \gamma_1)$ and $SDRS_2 = (\mathcal{N}_2, \Omega_2, \gamma_2)$ be the Subset Difference Revocation Schemes defined on the two subtrees of $T$ rooted at the left and right child of the root of $T$, $T_1$ and $T_2$. Let $DURS$ be the disjoint union of $SDRS_1$ and $SDRS_2$. Let $SDRS = (\mathcal{N}, \Omega_3, \gamma_3)$ be a Subset Difference Revocation Scheme with $2n$ users, defined on $T$. If $\mathcal{R} \neq \emptyset$ is a subset of revoked users corresponding to leaves which are descended from the same grandchild of the root of $T$ then:

$$t^{SDRS}(\mathcal{N}, \mathcal{R}) = t^{DURS}(\mathcal{N}, \mathcal{R}) - 1.$$

Otherwise, $t^{SDRS}(\mathcal{N}, \mathcal{R}) = t^{DURS}(\mathcal{N}, \mathcal{R})$.

*Proof.* From the definition of a disjoint union (Definition 48), the following subsets are in $DURS$:

$$\begin{aligned} \{\gamma(i,j) : \gamma(i,j) \in DURS\} &= \{\gamma_1(i,j) : \gamma_1(i,j) \in SDRS_1\} \\ &\quad \cup \{\gamma_2(i,j) : \gamma_2(i,j) \in SDRS_2\} \\ &\quad \cup \{\gamma(0,0)(= \mathcal{N})\}. \end{aligned}$$

Clearly $\gamma(i,j)$ is in $DURS$ and $SDRS$ when $(i,j) = (0,0)$ or when $i$ is in either $T_1$ or $T_2$, i.e. not the root. But there are also two other occasions when $\gamma(i,j)$ is in both schemes and $i$ is the root of $T$ (which is not in either of the two subtrees). When $j = right\_child(root)$, then:

$$\begin{aligned} \gamma(i,j) &= desc(root) \setminus desc(right\_child(root)) \\ &= desc(left\_child(root) \\ &= \gamma_1(0,0) \in SDRS_1 \subset DURS. \end{aligned}$$

The same holds when $j$ is the left child of the root. So, in order for a subset to be in $SDRS$, but not in $DURS$, it must be in the form of $\gamma(i,j)$ where $i$ is the root of $T$ and $j$ is at least distance two from the root. For example, if $j$ is in the right tree, then:

$$\begin{aligned}\gamma(i,j) &= desc(root) \setminus desc(j) \\ &= desc(left\_child(root)) \cup (desc(right\_child(root) \setminus desc(j)) \\ &= \gamma_1(0,0) \cup \gamma_2(\text{root of } T_2, j).\end{aligned}$$

Subsets of this form are composed of users from both $\mathcal{N}_1$ and $\mathcal{N}_2$. Since the only subset in $DURS$ with users from both $\mathcal{N}_1$ and $\mathcal{N}_2$ is $\gamma(0,0) = \mathcal{N}$ and there is at least one user not in $\gamma(i,j)$ ($desc(j)$), $\gamma(i,j) \notin DURS$. However, because any $\gamma(i,j)$ in this form is the union of $\gamma_b(i,j)$ and $\mathcal{N}_{1-b}$, for some $b \in \{0,1\}$, each subset in $SDRS$ that is not in $DURS$ can be written as the union of two subset that are in $DURS$. Therefore, if we let $v_1$ and $v_2$ be the roots of $T_1$ and $T_2$ respectively, we have:

$$\begin{aligned}\{\gamma(i,j) : \gamma(i,j) \in SDRS\} &= \{\gamma_1(i,j) : \gamma_1(i,j) \in SDRS_1\} \\ &\quad \cup \{\gamma_2(i,j) : \gamma_2(i,j) \in SDRS_2\} \\ &\quad \cup \{\mathcal{N}_1 \cup \gamma_2(i,j) : (i,j) \in SDRS_2, i = v_2\} \\ &\quad \cup \{\mathcal{N}_2 \cup \gamma_1(i,j) : (i,j) \in SDRS_1, i = v_1\} \\ &\quad \cup \{\gamma(0,0)(= \mathcal{N})\}.\end{aligned}$$

Clearly, $\{\gamma(i,j) : \gamma(i,j) \in DURS\} \subseteq \{\gamma(i,j) : \gamma(i,j) \in SDRS\}$, as all the subsets in $DURS$ are replicated in $SDRS$. Therefore, $t^{SDRS}(\mathcal{N},\mathcal{R}) \leq t^{DURS}(\mathcal{N},\mathcal{R})$, the minimal cover of $\mathcal{N} \setminus \mathcal{R}$ in $DURS$ is comprised of subsets that are also in $SDRS$.

Consider the minimal cover of $\mathcal{N} \setminus \mathcal{R}$ in $SDRS$. If this cover is comprised of subsets that are also in $DURS$, then $t^{SDRS}(\mathcal{N},\mathcal{R}) = t^{DURS}(\mathcal{N},\mathcal{R})$. Otherwise, there must be at least one subset in $SDRS$ that is not in $DURS$. From comparing the two lists of subsets in both schemes, any such subset must be in the form $\gamma(i,j)$ where $i$ is the root of $T$ and $j$ is distance 2 or further from the root. The privileged users in this subset are at the very least, all the descendants of one child of the root of $T$, and all the descendants of one

grandchild of the root. Therefore, all revoked users must be limited to one grandchild of the root.

Because there are more than half of the users in $\mathcal{N}$ in such a subset (all users are descended from $i$, 1/4 or less are descended from $j$), two subsets of this form would intersect. But by Corollary 71, no subsets in a minimal cover can intersect. Therefore there is only one such subset $\gamma(i,j) \in SDRS$ in the cover of $\mathcal{N} \setminus \mathcal{R}$ that is not in $DURS$. Because the subset can be written as the union of two subsets that are in $DURS$ ($\gamma(i,j) = \mathcal{N}_1 \cup \gamma_2(v_2,j)$ or $\mathcal{N}_2 \cup \gamma_1(v_1,j)$), we have:

$$t^{SDRS}(\mathcal{N},\mathcal{R}) = t^{DURS}(\mathcal{N},\mathcal{R}) - 1. \qquad \square$$

We now know exactly when $t(\mathcal{N},\mathcal{R})$ with the Subset Difference differs from the Disjoint Union, and by how much (if there is any difference, it is always 1). This leads to a very simple modification to Formula (5.5) for the average bandwidth of the Subset Difference Revocation Scheme.

*Theorem* 100. Let $SDRS$ be a Subset Difference Revocation Scheme with $2n$ users, defined in tree $T$. Then $SDRS$ has:

$$t^{SDRS}_{aver}(2n,r) = \left[ \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \frac{\binom{n}{r_1}\binom{n}{r-r_1}\left(t^{SDRS}_{aver}(n,r_1) + t^{SRDS}_{aver}(n,r-r_1)\right)}{\binom{2n}{r}} \right] - \frac{4\binom{\frac{n}{2}}{r}}{\binom{2n}{r}}$$

if $1 \le r \le n/2$. Otherwise:

$$t^{SDRS}_{aver}(2n,r) = \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \frac{\binom{n}{r_1}\binom{n}{r-r_1}\left(t^{SDRS}_{aver}(n,r_1) + t^{SRDS}_{aver}(n,r-r_1)\right)}{\binom{2n}{r}}.$$

*Proof.* Let $DURS$ be the Disjoint Union of two Subset Difference Revocation Schemes defined on the complete binary trees rooted at the left and right children of the root of $T$. By Lemma 99 we must have either $t^{SDRS}(\mathcal{N},\mathcal{R}) = t^{DURS}(\mathcal{N},\mathcal{R})$ or $t^{SDRS}(\mathcal{N},\mathcal{R}) = t^{DURS}(\mathcal{N},\mathcal{R}) - 1$. The latter only occurs when $\mathcal{R}$ is contained in the set of users corresponding to the descendants of one grandchild of the root of $T$. This is at most one quarter of the leaves of the tree. Consequently, if there is a difference in the size of the minimal cover in $SDRS$ and $DURS$ then $|\mathcal{R}| \le n/2$. There are 4 different grandchildren, and for each grandchild of the root there are $\binom{n/2}{r}$ subsets $\mathcal{R}$, where $|\mathcal{R}| = r$.

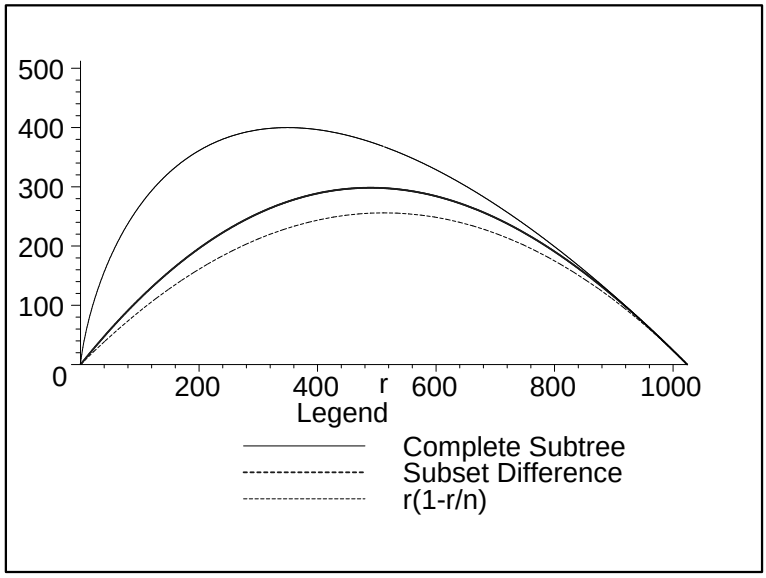


Figure 6.7: $t_{aver}(n, r)$ for the Complete Subtree and Subset Difference Revocation Schemes, for $n = 2^{10}$.

This gives a total of $4\binom{n/2}{r}$ subsets for which there is a difference in the size of the minimal cover in $SDRS$ and $DURS$. Therefore:

$$t_{aver}^{SDRS}(2n, r) = \begin{cases} t_{aver}^{DURS}(2n, r) - \frac{4\binom{n/2}{r}}{\binom{2n}{r}} \text{ if } 1 \leq r \leq n/2 \\ t_{aver}^{DURS}(2n, r) \text{ if } r > n/2. \end{cases}$$

Formula (6.10) gives a formula for $t_{aver}^{DURS}(2n, r)$ in terms of $t_{aver}^{SDRS}(n, r_1)$, for $r_1$ in the range $[\max(0, r-n), \ldots, \min(n, r)]$. Substituting this into the above gives us:

$$t_{aver}^{SDRS}(2n, r) = \left[ \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \frac{\binom{n}{r_1}\binom{n}{r-r_1}\left(t_{aver}^{SDRS}(n, r_1) + t_{aver}^{SRDS}(n, r-r_1)\right)}{\binom{2n}{r}} \right] - \frac{4\binom{\frac{n}{2}}{r}}{\binom{2n}{r}} \tag{6.11}$$

if $1 \leq r \leq n/2$, otherwise:

$$t_{aver}^{SDRS}(2n, r) = \sum_{r_1=\max(0,r-n)}^{\min(n,r)} \frac{\binom{n}{r_1}\binom{n}{r-r_1}\left(t_{aver}^{SDRS}(n, r_1) + t_{aver}^{SRDS}(n, r-r_1)\right)}{\binom{2n}{r}}. \tag{6.12}$$

□

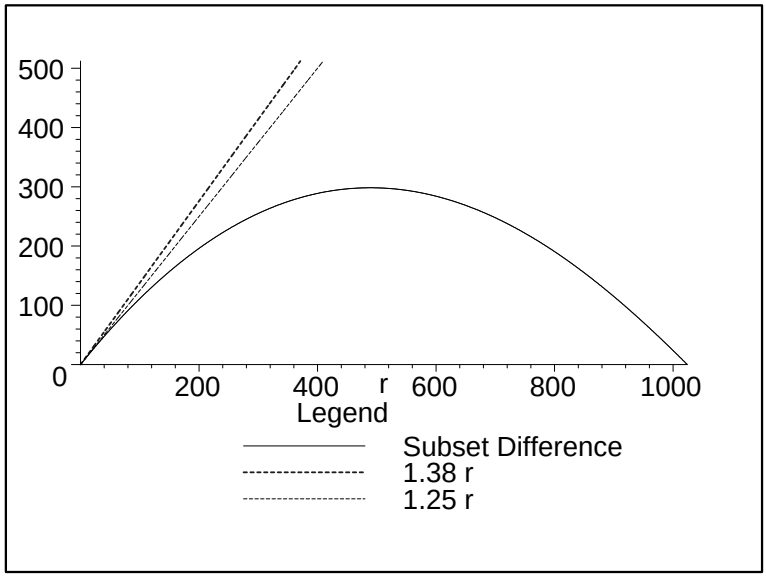


Figure 6.8: $t_{aver}(n, r)$ for the Subset Difference Revocation Schemes versus $1.38r$ and $1.25r$, for $n = 2^{10}$.

Figure 6.7 compares $t_{aver}(n, r)$ for the Complete Subtree and Subset Difference Revocation Schemes. This amounts to comparing the average bandwidth costs for the two schemes. We see that the Subset Difference Revocation Scheme gives a marked improvement over the Complete Subtree Revocation Scheme, most notably for small values of $r$. There is a large gap between the two graphs up until around $r = n/2$. In this range $t_{aver}(n, r)$ for the Subset Difference stays below 300, but that of the Complete Subtree hovers around the 400. When we compared $t_{\max}(n, r)$ of the two schemes (Figure 6.6), there was much less of a disparity, the two curves meeting at $r = n/2$. This implies that the difference in the bandwidth costs between the Subset Difference Revocation Scheme and the Complete Subtree Revocation Scheme is greater than that suggested just by $t_{\max}(n, r)$.

The fact that there is any difference between the two for $r \leq n/2 = 512$ also sheds more light on the comparison than the graphs for $t_{\max}(n, r)$ did. The maximums for both schemes were identical for all $r \geq n/2$, namely $t_{\max}(n, r) = n - r$. The average show that the Subset Difference performs better (even if it is only slightly) in this range.

The shape of $t_{aver}(n, r)$ for the two schemes is also different. The average

for the Complete Subtree peaks when $r \approx 3n/8$, while curve for the Subset Difference is almost symmetric, peaking near $r = n/2$. The most simple approximation of the latter is $r(n-r)/n$, while the former is close to $r \log_2(n/r)$. Of course, $r(n-r)/n$ is strictly below the curve of $t_{aver}(n,r)$ for the Subset Difference, while $r \log_2(n/r)$ is above that of the Complete Subtree.

In [23], Naor et al. proved that $t_{aver}(n,r)$ is bounded above by $1.38r$. They also conjectured $1.25r$ as a tighter bound, suggested by experimental evidence. We can see in Figure 6.8 that both of these are very pessimistic bounds. They certainly would not apply for $r \geq n/2$ where we know the maximum $t(\mathcal{N}, \mathcal{R})$ is $n - r$.

In this Chapter, we have established the exact formula for $t_{\max}(n,r)$ for the Subset Difference Revocation Scheme. This is a substantial improvement on the existing bound of $2r - 1$. It will also be important in comparing the performances of the schemes in the following Chapter. We have also given a recursive relation for $t_{aver}(n,r)$ which can be used for large values of $n$ (around 1000). We have already made some statements on how the Subset Difference Revocation Scheme compares to the Complete Subtree Revocation Scheme in terms of these two bandwidth measures. In Chapter 7, we will make more thorough comparisons.

# Chapter 7

# Comparison of Schemes

In the previous chapters we have looked at several existing Revocation Schemes, as well as creating new ones. Up to now the schemes have been largely judged either on their own merits, or compared to the original Complete Subtree Revocation Scheme. In this chapter we will try to make meaningful comparisons between all of the schemes. The goal is not to state explicitly which scheme is the "overall best", but rather to describe how a centre might go about deciding which scheme best satisfies the constrains for a particular scenario. The two main measures of performance we will use will be $t(\mathcal{N}, \mathcal{R})$ (in the form of both $t_{\max}(n, r)$ and $t_{aver}(n, r)$) and $|U|_{\max}$. The formulae for these will be quoted directly in this chapter with reference to their derivation.

There are five sections in this chapter. In the first, we do an example comparison of three schemes for one value of $n$. This highlights the difficulty in comparing the bandwidth costs of different schemes. In the next two sections we propose some metrics to quantify these costs and apply them to the schemes we have looked at. We then describe the complete process a centre would need to perform in order to determine the optimal scheme. Finally, we draw some overall conclusions.

## 7.1 Simple Comparison of Three Schemes

Let us start with a simple comparison of three schemes. If we fix the population size to be $n = 2^6 = 64$, then this specific case should give us some tools

| Scheme | Storage | $\|U\|_{\max}$ | Equation |
|---|---|---|---|
| $CSRS_2$ | $\log_2(n)+1$ | 7 | Equation (3.1) |
| $CSRS_4$ | $(2^3-1)\log_4(n)+1$ | 22 | Equation (3.4) |
| $SDRS$ | $\frac{1}{2}\log_2^2(n)+\frac{1}{2}\log_2(n)+1$ | 22 | Equation (3.5) |

Table 7.1: $|U|_{\max}$ for three different schemes when ($n = 64$).

for comparing all the schemes for different values of $n$. We will just look at the Complete Subtree Revocation Scheme on a binary tree ($CSRS_2$) and on a quaternary tree ($CSRS_4$) and the Subset Difference Revocation Scheme ($SDRS$). We choose not to consider a ternary tree because of the difference between $n$ and 81, the nearest power of 3. Any of the Forest of Trees schemes could also be used, but we will just consider the three above for now. We will first look at the storage, and then the bandwidth, of the three schemes. Finally, we will try to devise a way to combine the two measures for an overall view.

### 7.1.1 Comparing Storage

The simplest way of comparing the schemes is by looking at the storage. We calculate how many establishment keys each user must store under the different schemes. The respective formulae and values are in Table 7.1.

We have taken a slight liberty with the formulae. For $SDRS$, Equation (3.5) counts the number of labels a user stores. These are the labels that are used to generate the establishment keys as described in Section 3.3.2. The figure for the total number of establishment keys is just short of $4n$. We allow this method for reducing the storage in $SDRS$ because it does not expand the stored material. With the proper choice of Pseudo Random Number Generator (PRNG), the labels will be exactly the same length as the establishment keys. This makes labels in this scheme and establishment keys in another scheme comparable. Conversely, we do not allow the compression methods for the Complete Subtree Revocation Schemes of Asano [1]. The Master Keys are calculated using an RSA modulus. In order for this to be secure, the modulus

(and consequently the Master Keys) must be much larger than the establishment keys. The present minimum for an RSA modulus is around 1024 bits, while establishment keys could be between 64 and 128 bits (these are keys for a symmetric block cipher).

We see from Table 7.1 that $CSRS_2$ requires the fewest keys. The other two schemes, $CSRS_4$ and $SDRS$, are equal. The use of the PRNG in $SDRS$ will require extra computation for the users, but this is a different kind of cost that is not being considered yet. The ordering of the schemes in terms of storage is straightforward, i.e. the scheme that requires the fewest keys is the best. Unfortunately, the same is not true when it comes to comparing the bandwidth of the schemes.

### 7.1.2 Comparing Bandwidth

Our first measure of the bandwidth costs is the maximum: $t_{\max}(n, r)$. However, this function does not return a single value, but a range of values for all $r \in [0, \ldots, n]$ that correspond to the size of the largest header of a broadcast where $r$ users are revoked. The bigger the header is, the more information the centre has to broadcast to the users and so the higher the bandwidth. Figure 7.1 plots the three different graphs of $t_{\max}(n, r)$ for the three different schemes.

From Figure 7.1, we can see that both $SDRS$ and $CSRS_4$ have consistently lower bandwidth than $CSRS_2$. This is unsurprising as both schemes were designed with this goal in mind. However, neither $SDRS$ nor $CSRS_4$ has consistently lower bandwidth than the other for all values of $r$. For the values of $r$ from $r = 7$ to $r = 47$, $t_{\max}^{CSRS_4}(n, r) < t_{\max}^{SDRS}(n, r)$. For values of $r$ lower than 7, $t_{\max}^{SDRS}(n, r)$ is lower than $t_{\max}^{CSRS_4}(n, r)$, and for $r \geq 48$ the two are the same. This makes it difficult to rate either scheme better than the other. $CSRS_4$ does have the lower bandwidth for the bulk of the range of $r$, with significantly lower bandwidth around $r = n/2$. However, it may be that the values of $r$ such that $SDRS$ is less costly are the important values, i.e. the centre may only expect a few revoked users. Alternatively, if the higher values of $r$ are the important ones, then we cannot distinguish between the two schemes at all.

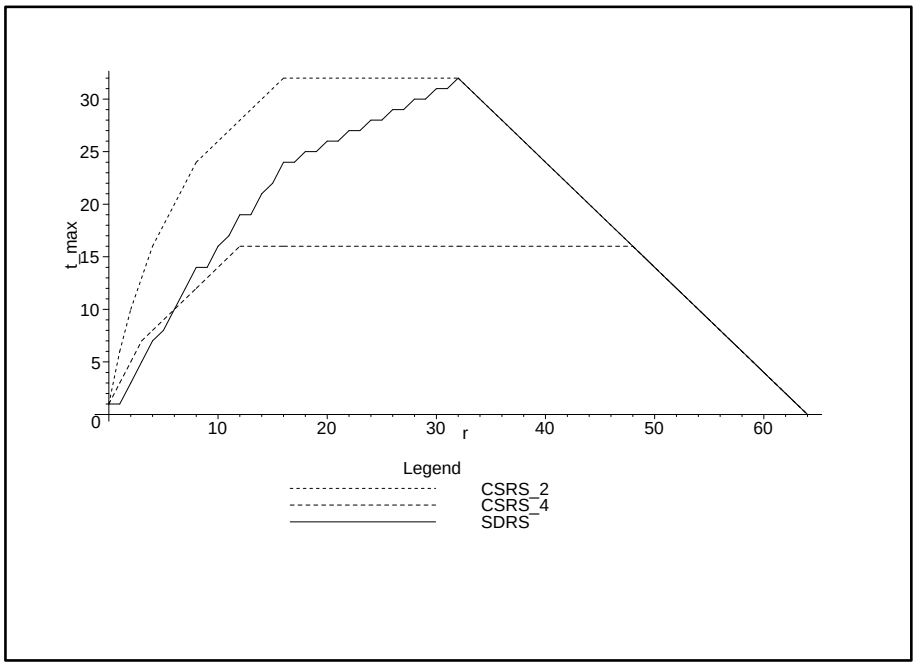


Figure 7.1: $t_{\max}(n, r)$ for three different Revocation Schemes

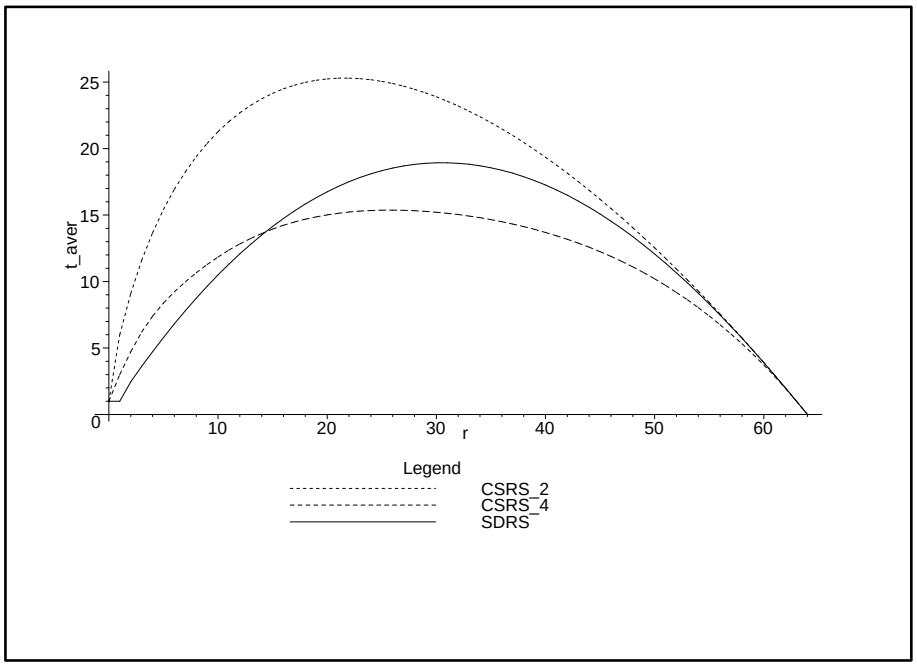


Figure 7.2: $t_{aver}(n, r)$ for three different Revocation Schemes

This last problem can be solved by plotting the average bandwidth cost: $t_{aver}(n, r)$, which is the average header size, averaged over all broadcasts when $r$ users are revoked. In Figure 7.2 we see $t_{aver}(n, r)$ plotted for $CSRS_2$, $CSRS_4$ and $SDRS$. The shapes of the graphs are different to those of $t_{\max}(n, r)$. We do not have jagged line segments but instead smooth curves. More importantly, for each value of $r$ there is a scheme with the lowest $t_{aver}(n, r)$ apart from the extreme values of $r = 0, n$. The only other point where two of the graphs meet does not occur at an integer value of $r$. So the graphs of $t_{aver}(n, r)$ allow us to better distinguish between the schemes than $t_{\max}(n, r)$ does. $CSRS_4$ has lower $t_{aver}(n, r)$ than $SDRS$ for large values of $r$, while the graphs of $t_{\max}(n, r)$ was equal for the two schemes for all $r \geq 3n/4$. Another observation we can make is that the average bandwidth for $CSRS_4$ and $SDRS$ is lower than that of $CSRS_2$ for all but the two extreme values of $r$. The maximum bandwidth for all three schemes was the same in the range $3n/4 \leq r \leq n$ (and for both $CSRS_2$ and $SDRS$ in the range $n/2 \leq r \leq 3n/4$). This shows that using either $CSRS_4$ or $SDRS$ will result in less bandwidth than $CSRS_2$ (on average).

The functions $t_{\max}(n, r)$ and $t_{aver}(n, r)$ give slightly different results when comparing schemes. While the graphs in Figures 7.1 and 7.2 are broadly similarly, the lines for $t_{aver}(n, r)$ do not overlap, but those of $t_{\max}(n, r)$ do, as has already been noted. Also, $t_{aver}(n, r)$ for $SDRS$ and $CSRS_4$ are considerably closer together than the respective values of $t_{\max}(n, r)$. We need to put these results in context, so we can say which results are more applicable. It could be argued that the maximum bandwidth is an artificial measure of bandwidth. For the most part, it is the size of a cover that has a vanishingly small probability of occurring. Consider the case when $r = n/2$. A set of revoked users that gives rise to the largest cover (in either $CSRS_2$ or $SDRS$) is any that has one user revoked out of every sibling pair. There are $2^{n/2}$ such sets out of a total $\binom{n}{n/2}$. In the case of $n = 2^6$ we get a probability of $2.34 \times 10^{-7}$, and this is a very small size for a scheme. When $n = 2^{10}$ the probability is just $2.99 \times 10^{-151}$.

There are advantages of using $t_{\max}(n, r)$ over $t_{aver}(n, r)$. For the most part it is easier to calculate, and in some cases it can be more pertinent. Suppose the centre will be delivering the broadcast some sort of storage media, e.g.

Compact Disks, DVD's, etc. In this case, the centre would need an upper bound on the number of ciphertexts in order to fix the area on the medium for the header. This would mean either finding the maximum $t_{\max}(n, r)$ over the range of all possible values of $r$, or finding the maximum over a smaller range, depending on the needs of the centre (i.e. how many revoked users the system must allow for).

One final point is that $t_{\max}(n, r)$ is not a consistent reflection of the average bandwidth. The shape of the plots of $t_{aver}(n, r)$ is roughly the same as that of the plots of $t_{\max}(n, r)$, although a plot of the average has a much more smooth curve. Naturally, the averages are lower than the maximums. However, the difference between the two varies for different schemes. For example, if we compare two graphs for $CSRS_2$ in Figure 7.1 and Figure 7.2 we see than $t_{\max}^{CSRS_2}(n, r)$ is only ever greater than $t_{aver}^{CSRS_2}(n, r)$ by at most 6.7 (when $r = 22$). However, $t_{\max}^{SDRS}(n, 32) - t_{aver}^{SDRS}(n, 32) = 12.1$. So any value of $t_{\max}(n, r)$ gives little information of the corresponding value of $t_{aver}(n, r)$ aside from being an upper bound. As the converse is true, we can not conclude that $t_{aver}(n, r)$ is a better measure of bandwidth. By the exact same argument, $t_{aver}(n, r)$ does not accurately reflect $t_{\max}(n, r)$. This does highlight the dangers of concentrating on only one of the two measures. In order to get the most well-rounded comparisons of schemes we will use both $t_{\max}(n, r)$ and $t_{aver}(n, r)$.

## 7.2 Proposed Bandwidth Scores

We are still left with the problem of deciding which scheme is more efficient in terms of bandwidth. We will describe some functions, or "scores", that take as argument $t(\mathcal{N}, \mathcal{R})$ (in the form of either $t_{\max}(n, r)$ or $t_{aver}(n, r)$) for a particular scheme as arguments and return a single value that reflects the bandwidth costs of that scheme. This value will then be used to rank the schemes in terms of bandwidth, just as $|U|_{\max}$ does for storages. The first score is:

$$score = \max_{r \in range_1} t_{\max}(n, r),$$

where $range_1$ is the expected range of $r$ in the proposed scheme. This measure would be appropriate for broadcasts on storage media as previously mentioned.

Knowing the maximum size of the header allows you to specify a fixed number of bits for the header. The fact that most broadcasts would not use all these bits does not matter, as the storage medium would have a fixed size. We could substitute $t_{aver}(n, r)$ for $t_{\max}(n, r)$ in the above score, but there is no obvious justification for the resulting score.

Even if the centre does not have information on the population's expected behavior, an estimate of the range $range_1$ can be made, based on the application. For example, in a Pay-per-View scheme, where users are given a wide range of content to choose from, one would expect the number of privileged users for any one broadcast to be small. Each user would only choose to view a small portion of the available content. If instead the application was a Subscription service, each user would be paying a flat rate. As such, we would expect the membership of the privileged set of users to be less volatile. In this case, it would be the number of revoked users that would be small. In the former case, the centre would set $range_1 = [(1-a).n, \ldots, n]$, and in the latter case, the centre would set $range_1 = [0, \ldots, a.n]$, where $0 < a \ll 1$ is a small fraction.

If the centre also knows the probability distribution of $r$, i.e. the probability that $r$ users will be revoked for any broadcast for $r \in range_1$, then we can define the score as:

$$score = \sum_{r \in range_1} P(r) \times t_{\max}(n, r) \quad \text{or} \quad score = \sum_{r \in range_1} P(r) \times t_{aver}(n, r).$$

This gives an expected value of either the maximum or average cover. We can be even more specific by defining the score using the probability of specific subsets $\mathcal{R}$ begin revoked:

$$score = \sum_{\mathcal{R} \subseteq \mathcal{N}} P(\mathcal{R}) \times t(\mathcal{N}, \mathcal{R}).$$

All of these scores are weighted averages that reflect the behavior or the population. However, they require knowledge of the probabilities of various revocation events. In the case of the latter score, the exponential number of subsets when $|\mathcal{N}|$ is large requires having an extensive list of probabilities, unless the vast majority were zero. This score would only be appropriate if there was a

| | $CSRS_2$ | $CSRS_4$ | $SDRS$ |
|---|---|---|---|
| $score_{\max}$ | 1366 | 820 | 1157 |
| $score_{aver}$ | 1100.66 | 715.86 | 801.45 |

Table 7.2: Scores for $CSRS_2$, $CSRS_4$ and $SDRS$

need to differentiate the probabilities of different sets of the same size being revoked. Otherwise, the average of $t_{aver}(n, r)$ with the appropriate probability distribution on $r$ will achieve the same result. The advantage of these scores is that they can reflect the fact that the most likely size of revoked subsets may be restricted to a small range. Assuming each user has the same probability of being revoked ($p$) then those subsets of revoked users with a size close to $n \times p$ will be the only ones with a non-negligible probability.

### 7.2.1 Application of Bandwidth Scores

In the absence of any specific probability distribution on $r$ or $\mathcal{R}$, we will use the following two scores:

$$score_{\max} = \sum_{r \in range_1} t_{\max}(n, r) \quad \text{and} \quad score_{aver} = \sum_{r \in range_1} t_{aver}(n, r).$$

These scores are easy to calculate given $t_{\max}(n, r)$ and $t_{aver}(n, r)$, but can be customised to different user behavior by varying $range_1$. We will be using the complete range, $range_1 = [0, \ldots, n]$, for most cases. This means that $score_{\max}$ and $score_{aver}$ are the areas under the graphs of $t_{\max}(n, r)$ and $t_{aver}(n, r)$ respectively. This would only be applicable if the centre needed the maximum resiliency in the scheme. For our three chosen schemes we get the scores in Table 7.2.

Table 7.2 confirms what we see from the graphs of $t_{\max}(n, r)$ and $t_{aver}(n, r)$, namely that $CSRS_4$ has the smallest area under the graph (for both graphs), followed by $SDRS$, and $CSRS_2$ has the greatest area. We will need the use of the scores later when comparing several schemes as it will not be readily apparent which schemes have the smallest area.

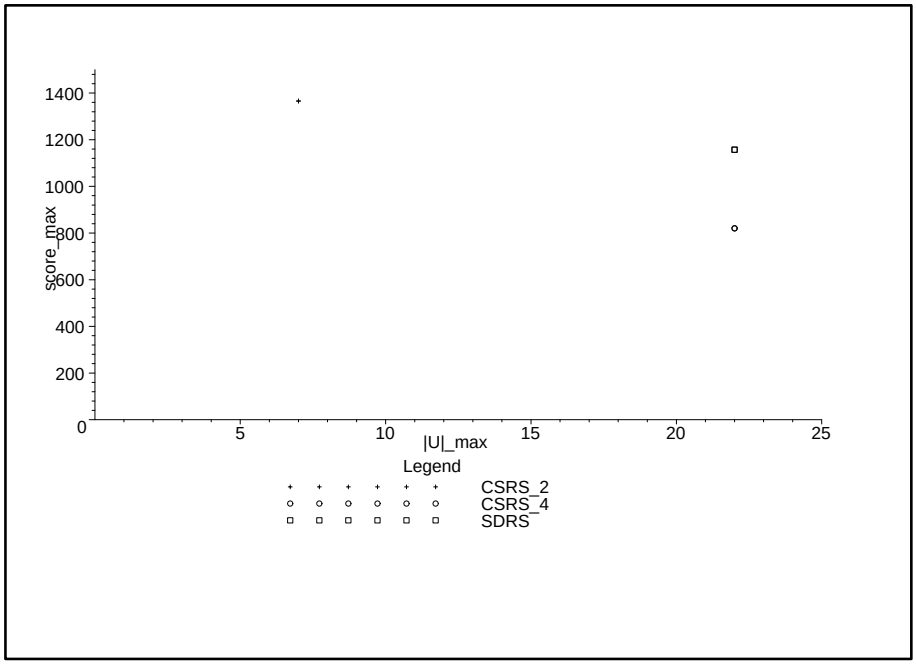


Figure 7.3: Combining figures for storage and maximum bandwidth

### 7.2.2 Combining Results

We have shown how to rate the storage of the three schemes, and a number of ways of rating the bandwidth as well. What we need now is an intuitive way of combining these measures, so the schemes can be judged with regard to both properties. The obvious way of doing this is to have a point on a plane for each scheme, where the $x$ co-ordinate is $|U|_{\max}$ and the $y$ co-ordinate is $score_{\max}$ (or $score_{aver}$). The closer a point is to the origin, the better the scheme is, i.e. the smaller $\sqrt{|U|^2_{\max} + score^2_{\max}}$ is, the better the scheme.

As the graph is drawn in Figure 7.3, the point for $CSRS_2$ appears closest to the origin. But because of the huge disparity in the axes ($y$ axis goes up to 1400, while the $x$ axis only goes as far as 22), it is the value of $score_{\max}$ that has the greatest influence on the distance to the origin. The point that is actually closest to the origin is that of $CSRS_4$, as a consequence of having the lowest $score_{\max}$. Because of the difference in the size of $score_{\max}$ and $|U|_{\max}$, $|U|_{\max}$ has almost no influence on which point is closest to the origin. In order to lessen the disparity, the $y$ axis should be scaled down by a constant factor. Ideally, the $score_{\max}$ values would be scaled down in such a way that the same

distance on each axis would represent the same cost. For example, if it was possible to translate both $|U|_{\max}$ and $score_{\max}$ into actual financial costs, then the graphs would represent the comparative costs more accurately.

It may not be possible to express the costs of both storage and bandwidth in similar terms, especially as storage would most likely be a one-time cost and bandwidth costs are repeated for each broadcast. However, there are two simpler ways for the centre to choose the best scheme for its needs. The first way is to decide the maximum possible storage that can be available to a user. The centre would then looks at all the points in Figure 7.3 with storage less than or equal to this value and picks the point with the lowest $score_{\max}$, or equivalently the point closest to the $x$-axis. For example, if the maximum storage was less than 22, then $CSRS_2$ would be chosen as it is the only scheme with less than 22 keys per user (unless the maximum was less than 7 as none of the three schemes has storage that low). If the maximum storage is any greater, then $CSRS_4$ would be chosen as it has the lowest $score_{\max}$ of the three. The other way is the same, except that the centre starts by limiting the bandwidth ($score_{\max}$) and choosing the scheme from those remaining that has the lowest storage.

Replacing $score_{\max}$ with $score_{aver}$ gives a similar plot: Figure 7.4. Since the order of $score_{aver}$ for the three schemes is the same as that for $score_{\max}$, there is no major difference. However, since $SDRS$ has a much smaller $score_{aver}$ than $score_{\max}$, it is closer to the origin (not as close as the point for $CSRS_4$ though).

A slight change to the range over which we calculate either score can give different results. If we change $range_1$ to be over small values of $r$, say $0, \ldots, n/4$, we get the scores in Table 7.3. For both $score_{\max}$ and $score_{aver}$, $SDRS$ scores lower than $CSRS_4$. The reason why $SDRS$ performs better is that $t_{\max}(n, r)$ is $2r-1$ for lower values of $r$, whereas with $CSRS_4$, $t_{\max}(n, r)$ is closer to $r \log_4(n/r)$. As this demonstrates, neither $score_{\max}$ nor $score_{aver}$ are absolute metrics for determining the best scheme. They only give a limited view of performance that depends on the chosen $range_1$.

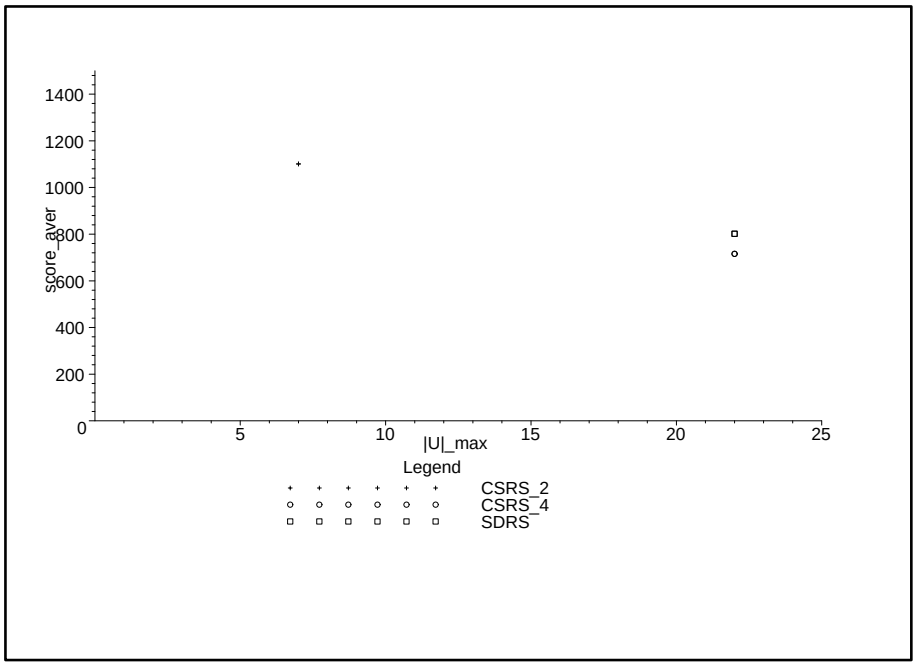


Figure 7.4: Combining figures for storage and average bandwidth

| | $CSRS_2$ | $CSRS_4$ | $SDRS$ |
|---|---|---|---|
| $score_{\max}$ | 155 | 79 | 75 |
| $score_{aver}$ | 131.74 | 71.82 | 51.59 |

Table 7.3: Scores for $CSRS_2$, $CSRS_4$ and $SDRS$ with smaller $range_1$

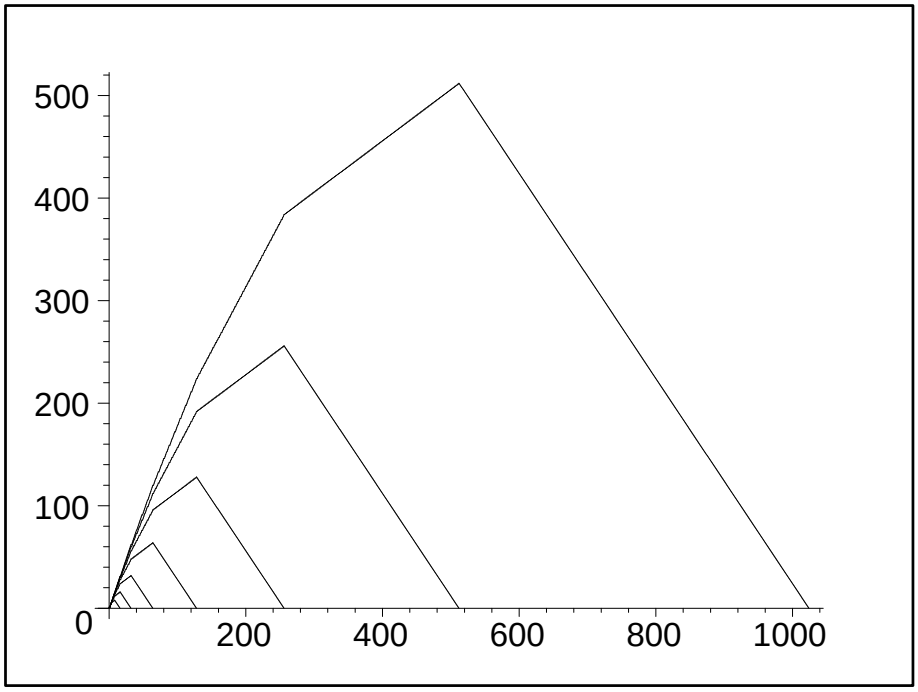


Figure 7.5: $t_{\max}(n, r)$ for $SDRS$ with $n = 2^4, \ldots, 2^{10}$

## 7.3 Comparing Schemes with different $n$

In the previous section we showed how a centre might choose between three schemes. But this was a very restricted example. The value of $n$ was fixed, and we only looked at three schemes. Granted, when choosing a scheme, a centre will probably know how many users will be in the system. However, we wish to know how all the schemes compare as $n$ grows. For the two bandwidth scores we used, $score_{\max}$ and $score_{aver}$, the Complete Subtree Revocation Scheme on a quaternary tree performed better than the Subset Difference Revocation Scheme when $n = 2^6$ (for one choice of $range_1$). We want to know if this is true for smaller and larger values of $n$. Also, we need to be able to compare schemes that have different size user sets, e.g. the number of users in a scheme based on a binary tree will be a power of 2, while the Complete Subtree Revocation Scheme on a ternary tree will have a power of 3 users. It turns out that both of these goals can be accomplished with the use of scaled down graphs.

Let us focus on how one Revocation Scheme, the Subset Difference Revocation Scheme, changes as $n$ grows. The storage grows in a very straight

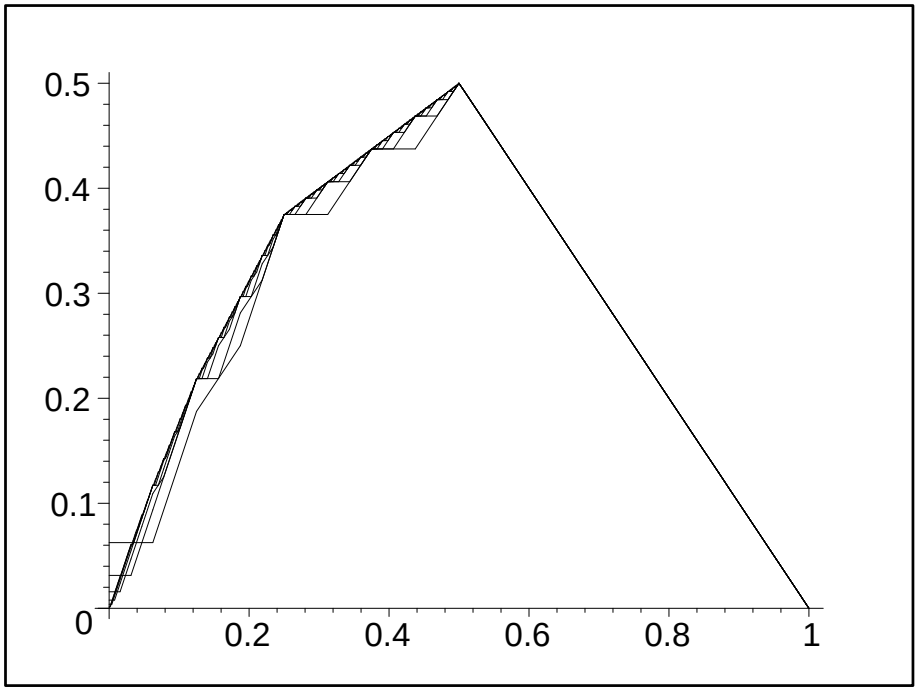


Figure 7.6: $t_{\max}(n,r)/n$ for $SDRS$ with $n = 2^4, \ldots, 2^{10}$

forward way: $|U|_{\max} = \frac{1}{2}\log_2^2(n) + \frac{1}{2}\log_2(n) + 1$. However, it is not obvious from the formula for $t_{\max}(n,r)$ (Formula (6.9)) how the bandwidth grows with $n$. In Figure 7.5 we plot $t_{\max}(n,r)$ on the same graph for $n$ equal to all powers of 2 from $2^4$ to $2^{10}$. We cannot directly compare any of the graphs as the range doubles from one to the next. If instead of plotting $[r, t_{\max}(n,r)]$, we plot $[r/n, t_{\max}(n,r)/n]$, then each graph will go from 0 to 1 in the horizontal axis and 0 to $\frac{1}{2}$ in the vertical, since $t_{\max}(n,r)$ for $SDRS$ (as well as for most revocation schemes) is bounded above by $\frac{n}{2}$. This is done in Figure 7.6.

This results in graphs that are close together, the many graphs overlapping making it difficult to distinguish individual ones. We can make some observations. The graphs seem to be increasing in area (as $n$ grows), but in smaller and smaller increments, tending to a limit. For example in the range $\frac{n}{4} \leq r \leq \frac{n}{2}$ we have a step function from $\frac{3n}{8}$ to $\frac{n}{2}$. When $n = 2^4$ we only get two "steps". But as $n$ grows we get more steps between the two points, making a closer approximation to a straight line. It is more complicated in the lower range, $1 \leq r \leq \frac{n}{4}$, but it does seem to tend to something close a straight line between $[0,0]$ and $[\frac{1}{4}, \frac{3}{8}]$. The only place where the graphs for smaller values

| $SDRS$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ | $2^8$ | $2^9$ | $2^{10}$ |
|---|---|---|---|---|---|---|---|
| $score'_{\max}$ | 0.2754 | 0.2798 | 0.2823 | 0.2840 | 0.2848 | 0.2852 | 0.2855 |
| $score'_{aver}$ | 0.2056 | 0.1988 | 0.1955 | 0.1940 | 0.1932 | 0.1923 | 0.1926 |
| $CSRS_2$ | $2^4$ | $2^5$ | $2^6$ | $2^7$ | $2^8$ | $2^9$ | $2^{10}$ |
| $score'_{\max}$ | 0.3360 | 0.3340 | 0.3335 | 0.3334 | 0.3333 | 0.3333 | 0.3333 |
| $score'_{aver}$ | 0.2824 | 0.2731 | 0.2687 | 0.2666 | 0.2655 | 0.2650 | 0.2648 |

Table 7.4: $score'_{\max}$ and $score'_{aver}$ for $SDRS$ and $CSRS_2$ with various $n$

of $n$ give the higher points is when $r = 0$. We always have $t_{\max}(0, n) = 1$, so the smaller $n$ is the less it will be scaled down.

By approximating the graphs to the implied straight lines, we can get an estimate of the area of the limiting case. The area under the lines connecting the points $[0, 0], [\frac{1}{4}, \frac{3}{8}], [\frac{1}{2}, \frac{1}{2}], [1, 0]$ is:

$$\left(\frac{1}{2} \times \frac{1}{4} \times \frac{3}{8}\right) + \left(\frac{1}{4} \times \frac{7}{16}\right) + \left(\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2}\right) = \frac{3}{64} + \frac{7}{64} + \frac{1}{8} = \frac{9}{32} = 0.28125.$$

With a minor modification to the formula, we can define a new score to be the area of scaled down graphs:

$$score'_{\max} = \sum_{r \in range_1} \frac{t_{\max}(n, r)}{n^2} \quad \text{and} \quad score'_{aver} = \sum_{r \in range_1} \frac{t_{aver}(n, r)}{n^2}.$$

The values we get for $range_1 = [0, \ldots, n]$ in Table 7.4, are all close to our estimate of 0.28125. But they are all above this value, and are tending to a limit that is a little above it (around 0.286). The reason for this is that $t_{\max}(n, r)$ (when scaled down) tends to a limit slightly above the straight line between $[0, 0]$ and $[\frac{1}{4}, \frac{3}{8}]$.

For the Complete Subtree Revocation Scheme on a binary tree, we can make a more definitive statement about $score'_{\max}$. By Theorem 102 in Appendix C we have that:

$$\sum_{r=0}^{n} t_{\max}(n, r) = \frac{1}{3}n^2 + \frac{2}{3}.$$

$$\text{Hence} \qquad \sum_{r=0}^{n} \frac{t_{\max}(n, r)}{n^2} = \frac{1}{3} + \frac{2}{3n^2}$$

$$\text{so that} \qquad score'_{\max} \to \frac{1}{3} \text{ as } n \text{ tends to infinity.}$$

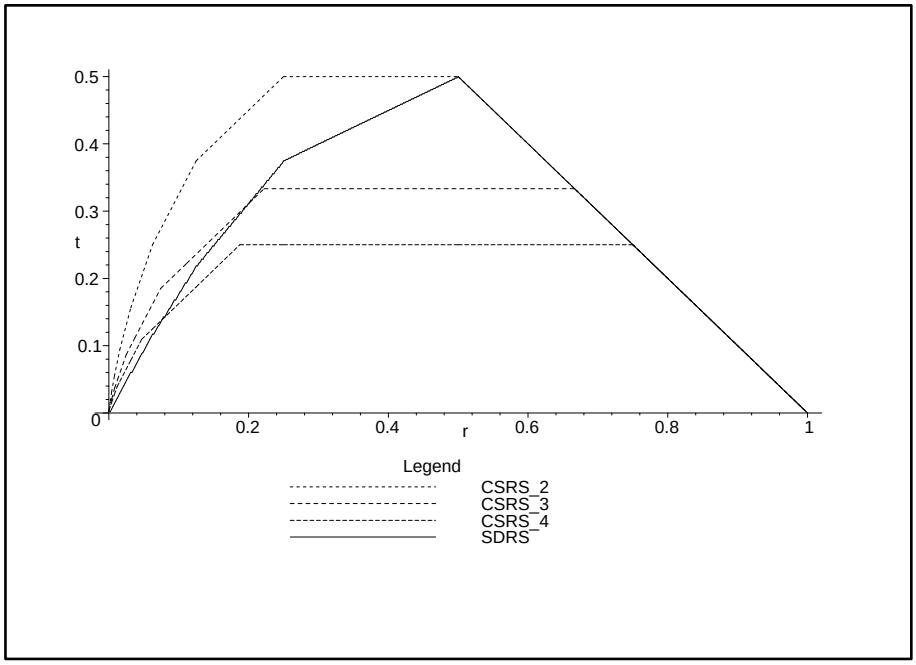


Figure 7.7: $t_{\max}(n,r)/n$ for $CSRS_2$ and $SDRS$ with $n = 2^9$, $CSRS_3$ with $n = 3^5$ and $CSRS_4$ with $n = 4^4$.

We use this to calculate $score'_{\max}$ for $CSRS_2$ in Table 7.4. While the values of $score'_{aver}$ for both $CSRS_2$ (and $SDRS$ in Table 7.4) also tend to a limit, due to the recursive nature of the formula for $t_{aver}(n,r)$, it is difficult to justify this for $n$ greater than the values we have calculated. For those values that we can calculate $score'_{\max}$ and $score'_{aver}$ for, we can see how the bandwidth changes as $n$ grows. We can also use this to compare the bandwidth of schemes with different sized user sets. When comparing the bandwidth of schemes, it is important to compare the storage at the same time, as is done if Figures 7.3 and 7.4.

In Figure 7.7, we have the graph of $r/n$ vs $t_{\max}(n,r)/n$ for four different schemes $CSRS_2$, $CSRS_3$, $CSRS_4$ and $SDRS$ with varying size user sets so the bandwidth can be compared directly. This adds the Complete Subtree Revocation Scheme on a ternary tree, which was absent from the earlier comparisons. Unsurprisingly, $t_{\max}(n,r)/n$ for $CSRS_3$ lies between $CSRS_2$ and $CSRS_4$. Just as was the case for $CSRS_4$, $CSRS_3$ has higher maximum bandwidth than $SDRS$ for small values of $r$, equal maximum bandwidth for large values of $r$ and lower maximum bandwidth for intermediate values of $r$.

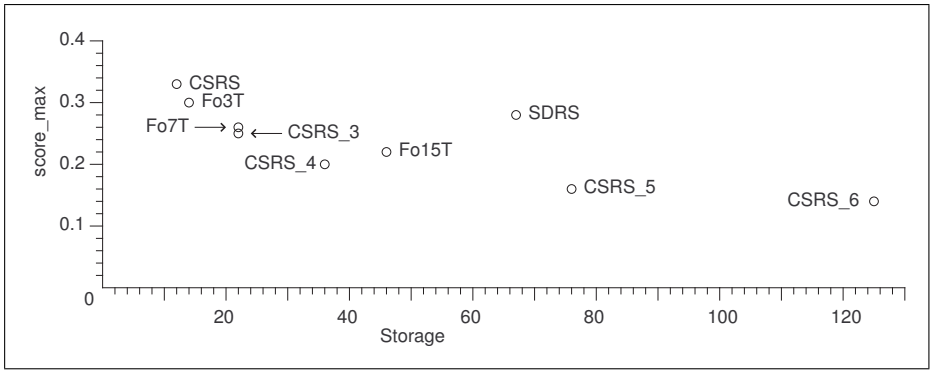


Figure 7.8: $|U|_{\max}$ versus $score'_{\max}$ for a variety of Revocation Schemes.

Fortunately, the use of $score'_{\max}$ removes the need for a dissection of several such graphs. In Figure 7.8 we have plotted $|U|_{\max}$ versus $score'_{\max}$ for nine different Revocation Schemes, including the four above. The values of $n$ for the different schemes are not the same by necessity (e.g. binary and ternary must have different size user sets) but has been kept to as small an interval as possible to give a good comparison. Each scheme is represented as a single point where the $x$ co-ordinate is $|U|_{\max}$, and the $y$ co-ordinate is $score'_{\max}$. The complete list of the schemes in the figure and their co-ordinates is:

**Binary Schemes:** $n = 2^{11} = 2048$. Complete Subtree ($CSRS_2 = [12, 0.33]$), Subset Difference ($SDRS = [67, 0.29]$), Forest of 3, 7 and 15 Trees ($Fo3T = [14, 0.30]$, $Fo7T = [22, 0.26]$ and $Fo15T = [46, 0.22]$).

**Ternary Scheme:** $n = 3^7 = 2187$. Complete Subtree on a ternary tree ($CSRS_3 = [22, 0.25]$).

**Quaternary Scheme:** $n = 4^5 = 1024$. Complete Subtree on a quaternary tree ($CSRS_4 = [36, 0.20]$).

**5-ary Scheme:** $n = 5^5 = 3125$. Complete Subtree on a 5-ary tree ($CSRS_5 = [76, 0.17]$).

**6-ary Scheme:** $n = 6^4 = 1296$. Complete Subtree on a 6-ary tree ($CSRS_6 = [125, 0.14]$).

The most striking feature of Figure 7.8 is how poorly the Subset Difference Revocation Scheme fairs. All of $CSRS_3$, $CSRS_4$, $Fo7T$ and $Fo15T$ have both

$score'_{\max}$ and $|U|_{\max}$ lower than that of $SDRS$. This is despite the fact that $SDRS$ only has the labels that generate the keys counting toward the storage. If we had used a smaller $range_1$ in calculating $score'_{\max}$, only summing over small values of $r$, we would get radically different results. $SDRS$ has the lowest $t_{\max}(n, r)$ for small values of $r$, and would consequently get the smallest $score'_{\max}$. Apart from the one errant point for $SDRS$ in the top-right of Figure 7.8, the rest of the schemes roughly transcribe a curve similar to that of $y \approx \frac{1.3}{\sqrt{x}}$. This curve marks the storage/bandwidth trade-off. The decision as to which scheme is most efficient is not clear-cut. Before this can be judged, the centre must work out the relative costs of bandwidth and storage. Once this is done, then one of the axes can be re-scaled so that the same distance on either axes represents the same cost. The best scheme is then the one with the point closets to the origin. If the relative costs are not comparable, then the centre can limit one cost and minimise the other (as was done for Figures 7.3 and 7.4).

It is worth noting that the Complete Subtree Revocation Schemes on the $a$-ary trees perform slightly better than the Forest of Trees Schemes. $CSRS_3$ is slightly below $Fo7T$ (same storage), while $CSRS_4$ is closer than $Fo15T$ to the origin along both axes.

Figure 7.9 is a more general plot than Figure 7.8. Each point is still in the form $[|U|_{\max}, score'_{\max}]$. But instead of one point for each scheme, we have a line of points marking the progress of the scheme as $n$ grows. The leftmost points represent $n = 2^4$, $3^2$, $4^2$, $5^2$ or $6^2$ as appropriate, and the rightmost points are the same as those in Figure 7.8, i.e. $n = 2^{11}$, $3^7$, $4^5$, $5^5$ or $6^4$. A centre could use such a plot to find the most appropriate scheme. By travelling along any one line until he reaches the first point that corresponds to a population size equal to or greater than the desired $n$, he will have the cost of storage and bandwidth for that particular scheme. By doing this for all lines in the plot, he will end up with something resembling Figure 7.8. The same techniques we mentioned for choosing the best scheme will still work: namely limiting the storage and choosing the scheme with the lowest bandwidth cost, or vice versa.

There is little variation in the heights of the points ($score'_{\max}$) along any

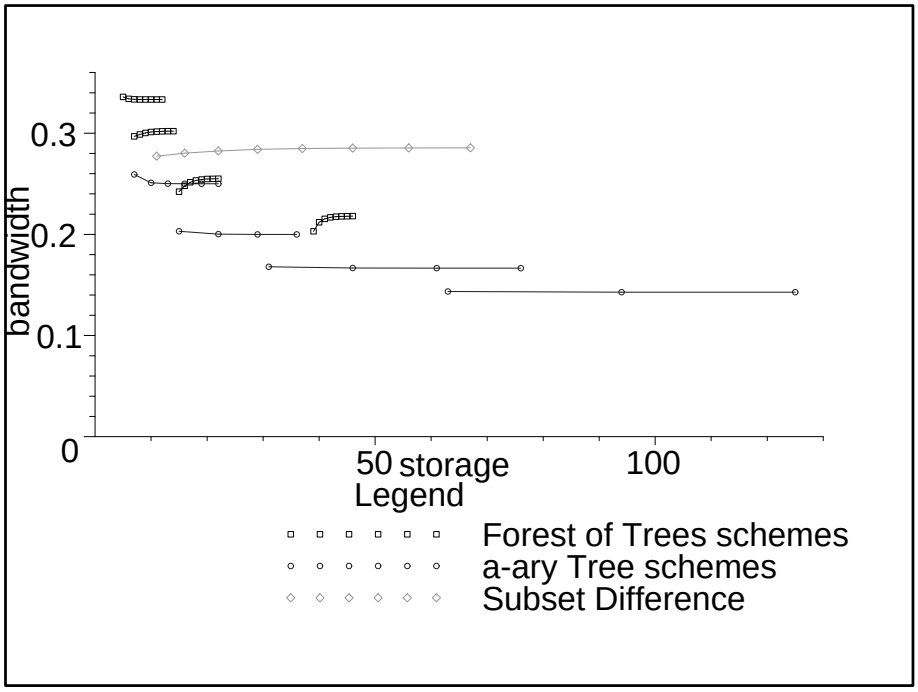


Figure 7.9: $|U|_{\max}$ versus $score'_{\max}$ for a variety of Revocation Schemes over different values of $n$.

one line, as each scheme seems to quickly converge to a fixed height either from above or below. We have already seen that $score'_{\max} \to 1/3$ from above in the case of the Complete Subtree Revocation Scheme. There is a greater variation in the way the storage grows with $n$ for the different schemes. This can be explained by looking at the three different groups of scheme we have: The Forest of Tree schemes, the $a$-ary tree based schemes and the Subset Difference Revocation Scheme.

For the Forest of Tree schemes we have the formula $|U|_{\max} = \log_2(n) + 1 + g(l-2) + 2$ (Formula (5.13)). The $g(l-2)+2$ term is constant for each scheme, taking the values $0, 2, 10$ and $34$ for $CSRS_2$, $Fo3T$, $Fo7T$ and $Fo15T$ respectively. This means:

$$\lim_{n\to\infty} \frac{|U|_{\max}}{\log_2(n)} = 1.$$

It also means that the storage increases by the same amount for each scheme as $n$ increases, and there is a constant offset between the storage of the different schemes. For example, for the same population $n$, the difference in $|U|_{\max}$ between $Fo15T$ and $Fo7T$ is always 24.

The corresponding formula for the Complete Subtree Revocation Scheme on an $a$-ary tree is:

$$\begin{aligned} |U|_{\max} = (2^{a-1}-1)\log_a(n)+1 &= \frac{2^{a-1}-1}{\log_2(a)}\log_2(n)+1 \\ \text{So} \quad \lim_{n\to\infty}\frac{|U|_{\max}}{\log_2(n)} &= \frac{2^{a-1}-1}{\log_2(a)}. \end{aligned}$$

The factor $(2^{a-1}-1)/\log_2(a)$ tells us how fast the storage grows, as it is constant for fixed $a$. For $CSRS$ on a binary tree (which technically falls into the categories of both Forest of Trees and $a$-ary tree scheme), the limit is 1. All larger values of $a$ give a higher limit: for 3-ary, 4-ary, 5-ary and 6-ary the limit is 1.89, 3.5, 6.46 and 11.99 respectively. In comparison, all storage for the Forest of Trees schemes is $\log_2(n)$ multiplied by 1, but with an added constant greater than 1. While some of the Forest of Trees schemes will have a higher storage than some $a$-ary tree schemes, the higher limit of $|U|_{\max}/\log_2(n)$ will mean that for a high enough $n$, any Forest of Trees scheme will have lower storage than any $a$-ary tree scheme. For example, $CSRS_3$ has lower storage than $Fo15T$ for all values of $n$ shown in Figure 7.9, but has greater storage when $\log_2(n) \geq 39$ or $n \geq 5.48 \times 10^{11}$. This is much to large for any practical scheme, being about 100 times the population of the planet!

The storage for the Subset Difference Revocation Scheme is unlike the other two types of scheme. For any line in Figure 7.9, aside from the one for $SDRS$, the points are evenly spaced. In the line for $SDRS$ we can see the distance between consecutive points increase as the line progresses (towards the right, i.e. as $n$ increases). All other schemes have storage $\mathcal{O}(\log(n))$, while $SDRS$ has $|U|_{\max} = 1/2\log_2^2(n) + 1/2\log_n^2(n) + 1$, which is $\mathcal{O}(\log^2(n))$ (so $|U|_{\max}/\log_2(n)$ does not converge). Because of this higher order of complexity, $SDRS$ will have the highest storage of these schemes as $n$ gets sufficiently large. For example, when $n = 2^{26}$ the storage for $SDRS$ is 352, but for $n = 6^{10}$ (nearest power of 6) the storage for $CSRS_6$ is 311. This is a very large value for $n$, but still within the realms of a actual scheme (it is less than the number of TV or Internet users in the US).

As was the case with the points in Figure 7.8, there is no clear "best" scheme. There are some general trends that we can discern. The $a$-ary

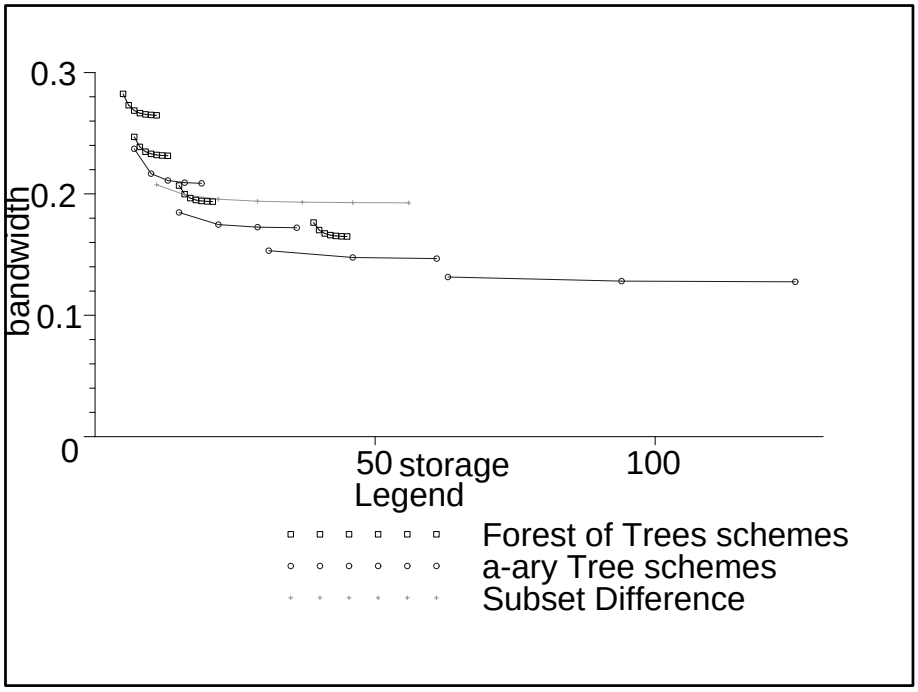


Figure 7.10: $|U|_{\max}$ versus $score'_{aver}$ for a variety of Revocation Schemes over different values of $n$.

tree schemes provide a lower $score'_{\max}$ than the Forest of Trees schemes do. The 5-ary scheme has almost half the $score'_{\max}$ of $CSRS$ (0.55), while the $Fo15T$ scheme has a $score'_{\max}$ greater than the quaternary tree scheme. Even though the $a$-ary schemes have (generally) lower storage than the Forest of Tree schemes for the parameter sizes in Figure 7.9, the above arguments mean that for larger values of $n$ that are more likely in an actual deployment of a Revocation Scheme, the Forest of Trees schemes would have lower storage. The Subset Difference Revocation Scheme appears to be the worst scheme in regards both $score'_{\max}$ and storage.

By plotting $|U|_{\max}$ versus $score'_{aver}$ we get different results, especially with regard to the Subset Difference Revocation Scheme. In Figure 7.10, we see that $SDRS$ has a $score'_{aver}$ between $CSRS_3$ and $CSRS_4$, which is roughly the same as $Fo7T$. This is much lower than $score'_{\max}$, both absolutely and relative to the scores for the other schemes. Like $score'_{\max}$, the values of $score'_{aver}$ quickly level off. Whereas for $score'_{\max}$ the limit was approached from above and below for different schemes, $score'_{aver}$ tends to a limit from above for all

schemes. The values of $|U|_{\max}$ are the exact same as in Figure 7.9, so all the statements made about the storage of the various schemes apply equally to Figure 7.10. Even though $SDRS$ has $|U|_{\max}$ and $score'_{aver}$ close to the schemes $CSRS_3$, $CSRS_4$ and $Fo7T$ for small values of $n$, because of the $\mathcal{O}(\log^2(n))$ storage, it grows much faster than the others. For larger values of $n$, $SDRS$ is one of the schemes furthest from the origin, having either a greater $|U|_{\max}$ or $score'_{aver}$ than all the other schemes.

There is a slightly different ordering of the other schemes given by $score'_{aver}$ than we had with $score'_{\max}$. $Fo7T$ has a $score'_{aver}$ that tends to a limit slightly below that of $CSRS_3$ (instead of above with $score'_{\max}$) and similarly, $Fo15T$ tends to a limit slightly below $CSRS_4$.

It is worth re-stating that these plots are examples of one particular way of calculating $score'_{\max}$ and $score'_{aver}$. Both scores are sums over $range_1$, which we set to $range_1 = [0, \ldots, n]$. It is more likely that the centre can place limits on how many revoked/privileged users there will be in the specific implementation of a scheme. The range of the sum can be correspondingly narrowed when calculating the scores. Even if the scheme must be set up to allow all possible values of $r$, then the centre will probably have some idea of the likelihood of the different values and could weight the different values in the sum accordingly.

Having said that, if we were to interpret Figures 7.9 and 7.10 as they are (i.e. for $range_1 = [0, \ldots, n]$), then the Subset Difference Revocation Scheme would have to one of the least efficient. The higher order storage quickly makes it more costly than the other schemes for all but the smallest values of $n$, and it does not give as low a bandwidth cost (of $score'_{\max}$ or $score'_{aver}$) as some of the other schemes. This is in stark contrast to conventional wisdom that says the Subset Difference Revocation Scheme is best of the current schemes ([16], [4], [2] and [8]). The reasoning for this is that the bound of $t_{\max}(n, r) \leq 2r - 1$ cannot be beaten by any other scheme for small values of $r$. Our analysis would suggest that the Complete Subtree Revocation Scheme on an $a$-ary tree gives the lowest overall bandwidth cost, but the most efficient in terms of bandwidth and storage is the Forest of Trees Revocation Schemes. The latter does not give as low a bandwidth score as the $a$-ary trees, but they do reduce the score from that of the Complete Subtree Revocation Scheme, and for much

less storage cost (at least for large values of $n$).

### 7.3.1 Comparing Compression Methods

In comparing the schemes in the last section, we ignored any benefit of using Compression Methods in the Complete Subtree Revocation Scheme on an $a$-ary tree. In order to present as fair a comparison of all the schemes as possible, we must look at the three methods described in Sections 3.2 and 4.3.3, and discover how much of a benefit they can be.

The main problem with analysing the Compression Methods is the fact that they are based on RSA calculations. Since each Master Key is essentially a number taken modulo an RSA modulus, the size is fixed, or at least bounded below by the minimum for a secure RSA modulus, e.g. around 1024 bits. We cannot directly compare these to the establishment keys for another scheme, as the bit length of the establishment keys is unspecified (but probably smaller than 1024).

Before we go any further, let us give a brief reminder of the function of the establishment keys. In the Revocation Protocol, described in Section 2.2, for any broadcast the message is encrypted under $E^1$ (a respected stream cipher) with a session key. This session key is encrypted several times under $E^2$ with different establishment keys. $E^2$ needs to be a secure block cipher, due to the small size of the input to the encryption function, as well as the requirement for the keys to be long lived. As far as analysing the storage of the Revocation Schemes, we are only interested in the key size of $E^2$. For the rest of this section, we will assume all establishment keys are 128 bits in length. This is a reasonable length given the requirements, i.e. it is the typical key length of current block ciphers (e.g. AES). We will also assume all RSA moduli to be 1024 bits in length. This allows us to compare the explicit storage of the various schemes.

In Figure 7.11 we have plotted the storage of the Complete Subtree Revocation Scheme on a binary tree using the various Compression Methods. The $x$-axis is $k$ or $\log_2(n)$ and goes up to $k = 30$ as $2^{30} = 1.07 \times 10^9$ is about the most users we would expect in a scheme. The $y$-axis is the storage required of the users in the schemes in kilobytes. There are five graphs in the figure,

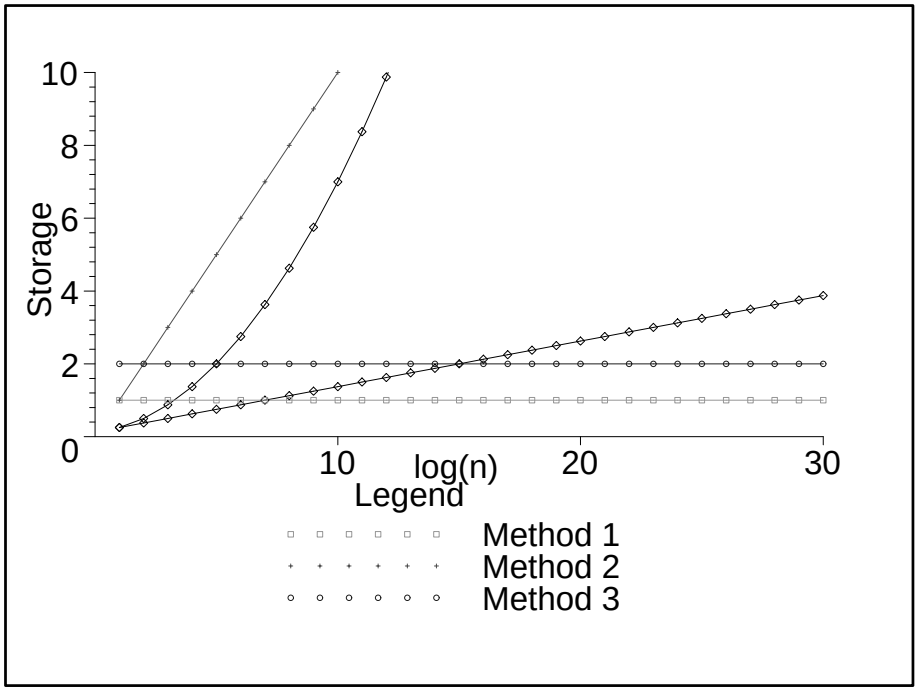


Figure 7.11: Storage (in kilobytes) versus $\log_2(n)$ for $CSRS_2$ (on a binary tree) with various Compression Methods.

although it is only the discrete points that represent values that would occur in a actual scheme. There is one line for $CSRS$ with the keys stored explicitly, and three for $CSRS$ using the three different Compression Methods. We also plot the storage for $SDRS$, which results in the only curve (storage is proportional to $\log^2(n)$). We will be using this curve in later comparisons. The storage of $CSRS_2$ using Method 1 represented by the completely horizontal line at $1KB$. This Compression Method only requires the storing of one Master Key, so it only depends on the size of the RSA modulus, and is completely independent of $a$ and $n$. For all but the smallest values of $k$, this requires less storage than storing the keys explicitly, as represented by the gently sloping line.

The second Compression Method is represented by the steeply sloped line that has, for the most part, the highest storage. An important observation is that the storage is consistently higher than that of $CSRS_2$ with the keys stored explicitly. The purpose of the compression methods is to reduce the amount of information stored by the users, but in this case Method 2 clearly fails.

Method 2 requires the user store $\log_a(n)$ Master Keys (in this case $a = 2$). But there are only $\log_2(n) + 1$ keys in the scheme and because Method 2 uses much larger keys, it has a higher cost. We can work out the minimal value of $a$ in order to have a saving in the storage with Method 2. For the general case, we will have $(2^{a-1} - 1)\log_a(n) + 1$ keys if they are stored explicitly. So Method 2 will provide a saving if:

$$\log_a(n) \times \text{RSA key} < ((\log_a(n))(2^{a-1} - 1) + 1) \times \text{AES key},$$

$$\text{that is if} \quad \log_a(n) \times \frac{\text{RSA key}}{\text{AES key}} < ((\log_a(n))(2^{a-1} - 1) + 1),$$

which is satisfied if:

$$\log_a(n) \times \frac{\text{RSA key}}{\text{AES key}} < ((\log_a(n))(2^{a-1} - 1)),$$

$$\text{i.e.} \qquad 8 < 2^{a-1} - 1,$$

$$\text{that is} \qquad a > 4.17.$$

So Method 2 provides actual compression on the stored information for $a \geq 5$ (for the given sizes of RSA and establishment keys). We can easily show that for any other values of $a$ ($a = 2,\ 3,\ 4$) we do not get any compression, assuming $\log_a(n) > 1$. For these values we have $2^{a-1} - 1 \leq 7$, so the explicit storage will be:

$$\begin{aligned}(\log_a(n)(2^{a-1} - 1) + 1) \times \text{AES key} &\leq (7\log_a(n) + 1) \times 128 \\ &< 8\log_a(n) \times 128 \\ &= 1024\log_a(n).\end{aligned}$$

For these values, the explicit storage is strictly less than that given by Method 2. So the only values of $a$ that provide compression are $a \geq 5$.

Like Method 1, Method 3 only requires that the user store a constant number of Master Keys for any scheme, namely $2^a - 2$. This gives us the second horizontal line in Figure 7.11, only this one is at $2KB$. For values of $\log_2(n) \geq 16$ this results in less storage than storing the keys explicitly. While it does have twice the storage of Method 1, there are several benefits of using Method 3 instead. This list of primes in Method 3 is much shorter

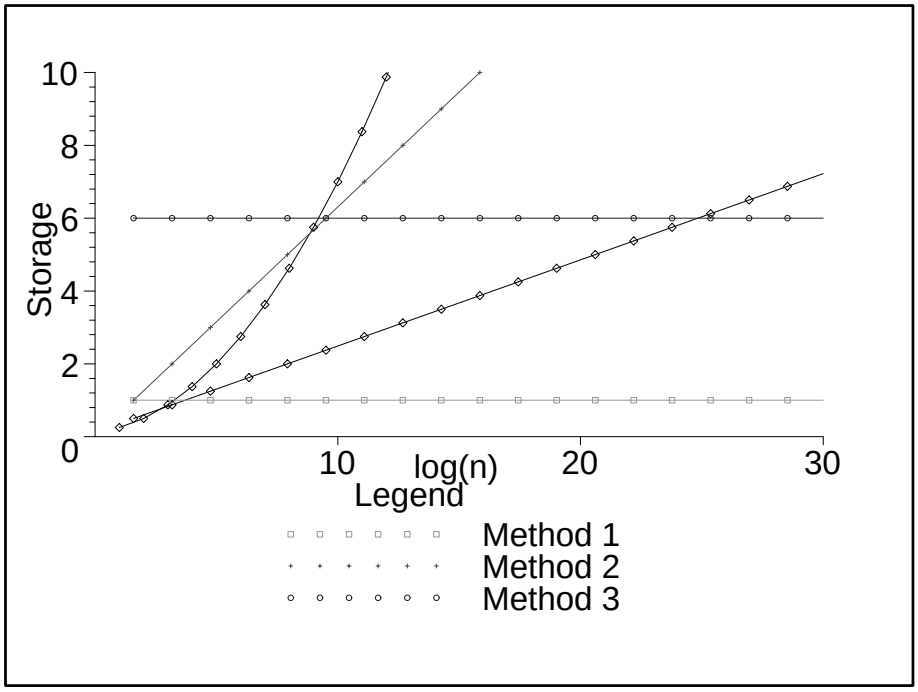


Figure 7.12: Storage (in kilobytes) versus $\log_2(n)$ for $CSRS_3$ (on a ternary tree) with various Compression Methods.

than that in Method 1 (and consequently the primes can be smaller). The operations are also much simpler: no prime generation needed, several small-sized exponentiations versus one very large-sized exponentiation.

Figure 7.12 shows the storage for the same Compression Methods, only with a Complete Subtree Revocation Scheme on a ternary tree. It is the same range of $\log_2(n)$ from 1 to 30, but because $n$ increases in powers of 3, the points are spaced further apart. Method 2 still requires more storage than just storing the keys explicitly, as does Method 3 for all but the largest values of $\log_2(n)$. Method 1 stays constant at $1KB$, clearly the best in terms of storage. In Figure 7.13 we jump up to $a = 5$. We can see how Method 2 reduces the storage from storing the keys explicitly, while Method 3 is consistently above both Method 2 and explicit storage.

In Chapter 4 we listed all the requirement of the various Compression Methods. These included multiplications and modular exponentiations, access to a public list of primes, and primality testing. What was left unsaid in these discussions was that when storing the keys explicitly there are no such

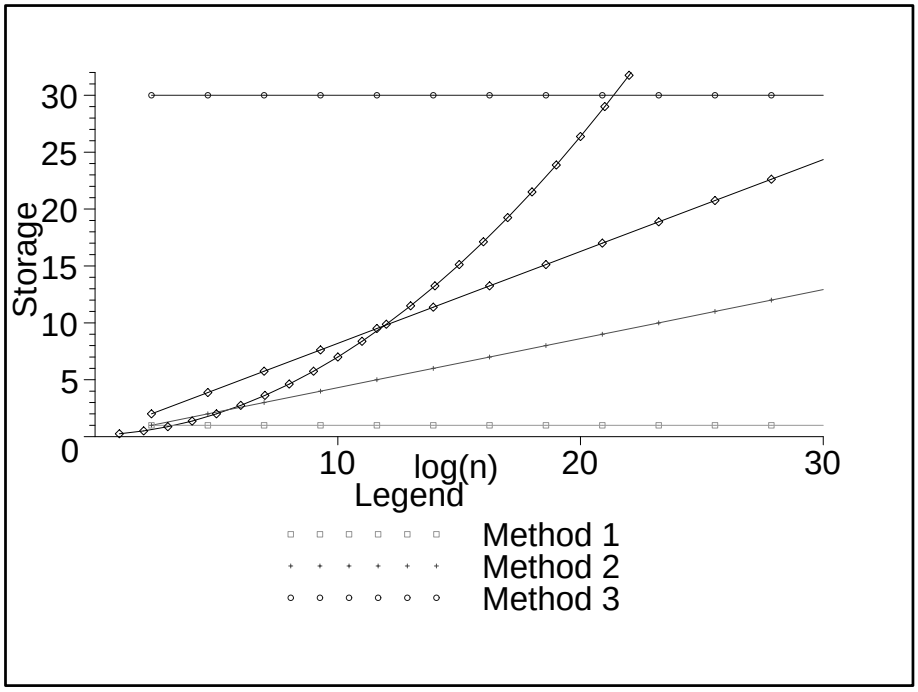


Figure 7.13: Storage (in kilobytes) versus $\log_2(n)$ for $CSRS_5$ (on a 5-ary tree) with various Compression Methods.

requirements. If in any one case Method 2 requires storing less bits than Method 3, then an argument could be made that because Method 3 is less computationally expensive, it could be the more desirable method. No such argument could be made if any Method requires more bits to be stored than storing the establishment keys explicitly, as there are no extra costs for the latter method. Therefore, for these key sizes (RSA and establishment) Method 2 should only be used when $a \geq 5$ and Method 3 for $a = 2$ and $n \geq 2^{16}$ or $a = 3$ and $n \geq 3^{16}$. For higher values of $a$, the value of $n$ at which Method 3 provides a storage saving is far too high. And for all but extremely small values of $n$, Method 1 provides a storage saving.

We also plotted the storage for the Subset Difference Revocation Scheme. The Compression Methods generally give lower storage than $SDRS$ in each of the figures. This is not very remarkable as the explicit storage of the keys is lower than $SDRS$ in most cases. The graphs do show the trend of the slope of the explicit storage; the line gets steeper as $a$ grows. Even at $a = 5$, the storage is less than that of $SDRS$ for just slightly more than half of the values

of $\log_2(n)$ shown. The explicit storage is $((2^{a-1}-1)\log_a(n)+1) \times |\text{AES key}|$, which is an order of complexity less than the $\approx 1/2\log_2^2(n) \times |\text{AES key}|$ of $SDRS$. This does mean that for $\log(n)$ large enough $SDRS$ will have the greater storage, but for large enough $a$ this would only be a scheme that is too big to ever occur in practice. Luckily, the slope of the storage for Method 2 follows the reverse trend. It gets more gradual as $a$ increases since it is plotting $\log_a(n) \times |\text{AES key}|$. So Method 2, and of course Method 1, will give lower storage than $SDRS$ for large values of $a$.

## 7.4 Strategy for choosing a scheme

This chapter has been concerned with the various ways of the Revocation Schemes we have looked at can be measured and rated. We will summarise these ideas in the form of a comprehensive strategy for a centre to decide what scheme to use for a particular application. It is assumed that the centre knows (or at least has a good approximation of) $n$, the number of users in the scheme, and has chosen appropriate encryption algorithms for $E^1$ and $E^2$.

The first step is to check if the computational capabilities of the receivers will be limited in a way that rules out certain schemes. If the receivers could not perform $RSA$ calculations, then the Complete Subtree on an $a$-ary tree would be possible, but none of the compression methods could be used. Even if $RSA$ calculations are possible, the processing power available will be finite. The costs of each compression method (Multiplications and Modular Exponentiations, as summarised in Section 4.3.3) will possibly rule out some of the compressions methods, and certainly limit which values of $a$ are possible. The cost of the storage of primes is another consideration, but separate from $|U|_{\max}$ as the primes are public. This cost can either be treated like the computation cost and used to limit the compression methods, or if any type of storage at the receivers is expensive, then it can be added to $|U|_{\max}$ to give the storage costs. The only other scheme that might be ruled out at this stage is the Subset Difference Revocation Scheme. If the receivers are not capable of executing the Pseudo-Random Sequence Generator, then the high costs of explicitly storing the keys would almost certainly make this scheme unfeasible.

The next step is for the centre to choose a bandwidth score. This could be any of the scores mentioned in Section 7.2, or a variant on these. The score may require knowledge of the expected behavior of the population, or the expected range of $r$. But even if this information is available, the most relevant score is not necessarily the one that uses all the information available, but the one that most closely reflects the costs of broadcasts. How well the score does this will have consequences in the final decision.

On choosing an appropriate score, the centre should calculate the points for a plot similar to Figure 7.8, for all the schemes that passed the first step. But rather than plotting the bandwidth score against $|U|_{\max}$, it should plot it against the explicit storage of either establishment keys (keys for $E^2$) or Master Keys/Labels in kilobytes. Any public primes can be added to this storage cost if appropriate.

Before the most apt scheme can be chosen, the costs of bandwidth and storage must be made comparable. If the bandwidth score the centre picked is an accurate measure of financial cost, and the storage axis can be re-labelled to give the storage costs, then the centre need only pick the scheme with a point closest to the origin. If this is not possible, then the centre would need to use the decision strategy mentioned earlier:

- Limit either the bandwidth score or storage, and remove all points above this limit on the appropriate axis
- Choose the remaining point that minimises the distance on the other axis.

By doing this, the centre either limits the allowable bandwidth cost and chooses the scheme with the minimum storage, or visa versa.

## 7.5 Conclusions

Most of the existing literature on the subject seems to point to the Subset Difference Revocation Scheme as being the best scheme. What we have done is shown that this is not a clear-cut decision, and arguments can be made for the other schemes. If we consider the performance of the schemes over

the entire range of possible values of $r$, as is done in Figures 7.9 and 7.10, then there are several schemes that perform better in terms of both storage and bandwidth. For example, the Forest of 15 Trees Scheme has values of both $score'_{\max}$ and $score'_{aver}$ lower than those of $SDRS$. Since its storage is $\log_2(n) + \mathcal{O}(1)$ compared to $\mathcal{O}(\log^2(n))$ it has lower storage for most values of $n$. As for the $a$-ary tree schemes, from $a \geq 4$ all schemes also have lower values of both $score'_{\max}$ and $score'_{aver}$ lower than those of $SDRS$ (ternary scheme only has a lower $score'_{\max}$). While the quaternary scheme does have storage lower than $SDRS$ for most values of $n$, for larger values of $a$ this is no longer true. But as we have shown in Section 7.3.1, for all $a \geq 5$, the second Compression Method for $a$-ary tree schemes costs less storage than storing the keys explicitly and less than the storage of labels in the $SDRS$ (see Figure 7.13).

The Subset Difference Revocation Scheme does have a lower bandwidth than any other scheme (with the exception of the trivial scheme in Lemma 7) for very small values of $r$. If the range over which $score'_{\max}$ and $score'_{aver}$ is calculated is narrowed to such values of $r$, then $SDRS$ would score better than any of the other schemes. However, as we see in Figures 7.1, 7.2 and 7.7, the bandwidth of the Complete Subtree Revocation Scheme on a quaternary tree is only slightly higher. Also, the interval for which the Subset Difference Scheme has the lowest bandwidth gets smaller when compared with schemes on $a$-ary trees with large $a$. This, coupled with the lower storage mentioned above, results in an $a$-ary tree scheme being a more efficient scheme that $SDRS$ with respect to both bandwidth and storage.

As we have said from the beginning, no scheme can be considered the "best" under all circumstances. A centre can only decide what scheme is most suitable to its particular needs. There can be many constraints, the ones we have discussed being:

- The capacity of information that can be broadcast by the centre (which either has a limit on the maximum or average size of a broadcast).
- The range of expected (or even likely) numbers of revoked users.
- The storage capacity at the receiver.

- The computational capability of the receiver.

We have shown how to analyse several schemes in these terms, and how to rate the best. The relative importance of how a scheme performs with regard the different properties (e.g. is storage more important than bandwidth?) is not something that can be stated in absolute terms. Once again, this is something that depends on the intended application, and needs to be decided by the centre. Once the centre knows its needs and priorities, we believe we have described the tools necessary to find the most applicable scheme.

# Appendix A

# Bound on $t_{\max}(n, r)$

The following Lemma is given a sketch proof in [23]. Using that fact that $t(\mathcal{N}, \mathcal{R})$ is just the number of nodes in $ST(\mathcal{R})$ whose degree is strictly less than its degree in the original tree we have:

*Lemma* 101. Let $(\mathcal{N}, \Omega, \gamma)$ be a Complete Subtree Revocation Scheme with $n = 2^k$ users. Then for all $1 \leq r \leq n$:

$$t_{\max}(n, r) \leq r \log_2(n/r).$$

*Proof.* Our induction hypothesis is that $t_{\max}(2^k, r) \leq r(k - \log_2(r))$, for any positive $r$, where $k$ is the height of the complete tree. The initial case is when $k = 1$. We only have two different values of $r$ to check, and by inspection we have $t_{\max}(2, 1) = 1$ and $t_{\max}(2, 2) = 0$. Substituting the respective values into the above formula we get $1 \log_2(2/1) = 1$ and $2 \log_2(2/2) = 0$, so the hypothesis holds.

Assume that the hypothesis is true for $k = i$, i.e. for a tree of height $i$, $t_{\max}(2^i, r)$ is bounded above by $r(i - \log_2(r))$. Then consider a tree of height $i + 1$. As a result of Lemma 13, $t_{\max}(n, r)$ is the maximum of the number of nodes in $ST(\mathcal{R})$ whose degree is less than their degree in the original tree, for $\mathcal{R}$ with $|\mathcal{R}| = r$. If all $r$ leaves are in one half of the tree, then $ST(\mathcal{R})$ is just the union of a path from the root to one of its children and a Steiner Tree $ST(\mathcal{R})'$ determined by $\mathcal{R}$ in the subtree of height $i$ whose root is that child. The root has degree in $ST(\mathcal{R})$ less than its degree in the original tree (it only has one child in $ST(\mathcal{R})$). Any other node will only have degree in $ST(\mathcal{R})$ less

that its degree in the original tree if that is the case in $ST(\mathcal{R})'$. Hence:

$$t_{\max}(2^{i+1}, r) \leq 1 + r(i - \log_2(r)) \leq r(i + 1 - \log_2(r)).$$

Otherwise, the leaves of $\mathcal{R}$ will be split between the two subtrees of height $i$ whose roots are the children of the root of the original tree, into a partition $\mathcal{R}_1$ and $\mathcal{R}_2$. A node has degree in $ST(\mathcal{R})$ less than its degree in the original tree if and only if it has degree in $ST(\mathcal{R}_i)$ less than its degree in its respective subtree corresponding to $\mathcal{R}_i$. Thus

$$t(\mathcal{N}, \mathcal{R}) \leq r_1(i - \log_2(r_1)) + r_2(i - \log_2(r_2))$$

where $r_1 = |\mathcal{R}_1|$ and $r_2 = |\mathcal{R}_2|$. Using the fact that $r_2 = r - r_1$, a little re-arranging gives us:

$$\begin{aligned} t(\mathcal{N}, \mathcal{R}) &\leq r_1(i - \log_2(r_1)) + r_2(i - \log_2(r_2)) \\ &= r.i - (r_1 \log_2(r_1) + r_2 \log_2(r_2)) \\ &= r.i - (r_1 \log_2(r_1) + (r - r_1) \log_2(r - r_1)). \end{aligned}$$

Define $f(r_1)$ to be this last expression. We need to show that this function is bounded above by $r(i + 1 - \log_2(r))$. We are only interested in this function in the range $1 \leq r_1 \leq r - 1$ (since both $r_1$ and $r_2$ are positive). From the derivative of $f(r_1)$ we can work out the maximum value in the given range.

$$\begin{aligned} f'(r_1) &= 0 - \frac{r_1}{r_1 \ln(2)} - \log_2(r_1) - \frac{-(r - r_1)}{(r - r_1)\ln(2)} - (-1)\log_2(r - r_1) \\ &= \log_2(r - r_1) - \log_2(r_1) \\ &= \log_2\left(\frac{r - r_1}{r_1}\right). \end{aligned}$$

The slope is only zero when $r - r_1 = r_1$ or $r_1 = r/2$. If $r_1 = r/2$, then we just get:

$$\begin{aligned} f(r/2) &= r.i - ((r/2)\log_2(r/2) + (r/2)\log_2(r/2)) \\ &= r.i - r(\log_2(r/2)) \\ &= r(i - (\log_2(r) - \log_2(2))) \\ &= r(i + 1 - \log_2(r)). \end{aligned}$$

In the range $1 \leq r_1 < r/2$, $f(r_1)$ is increasing (log of a number greater than 1 is positive), and in the range $r/2 < r_1 \leq r-1$ it is decreasing (log of a number less than 1 is negative). All this means that the maximum value of $f(r_1)$ in the range $1 \leq r_1 \leq r-1$ is $f(r/2) = r(i+1-\log_2(r))$. Because $t(\mathcal{N}, \mathcal{R})$ is bounded by $f(r_1)$, for all $r_1$, we have that $t_{\max}(2^{i+1}, r)$ is also bounded by $r(i+1-\log_2(r))$. This proves the induction hypothesis:

$$\begin{aligned} t_{\max}(n, r) &\leq r(k - \log_2(r)) \\ &= r(\log_2(n) - \log_2(r)) \\ &= r(\log_2(n/r)). \end{aligned}$$

□

# Appendix B

# Examples of $t_{\max}(n, r)$

Here we have some examples of $ST(\mathcal{R})$ when $t(\mathcal{N}, \mathcal{R}) = t_{\max}(n, r)$ for the Complete Subtree Revocation Scheme.

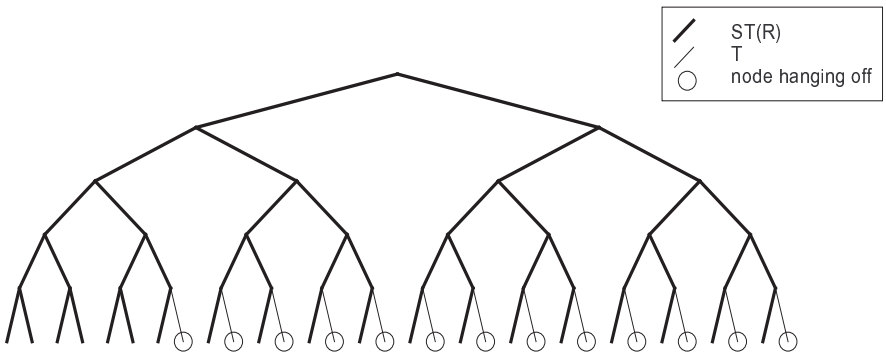


Figure B.1: $n = 2^5$, $r = 19$, $t_{\max}(n, r) = 13$

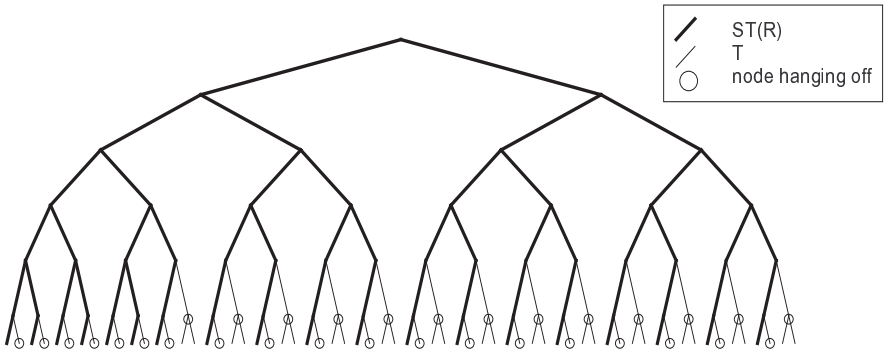


Figure B.2: $n = 2^6$, $r = 19$, $t_{\max}(n, r) = 32$

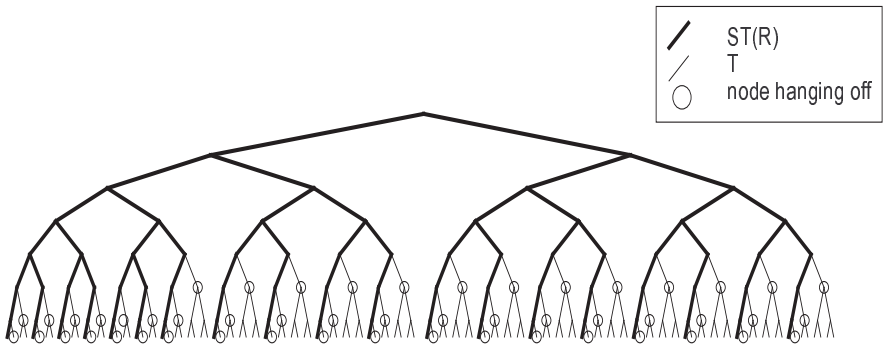


Figure B.3: $n = 2^7$, $r = 19$, $t_{\max}(n, r) = 51$

# Appendix C

# Summation of $t_{\max}(n, r)$

In this appendix, we prove the following result:

*Theorem* 102. Let $CSRS$ be a Complete Subtree Revocation Scheme on a binary tree with $n = 2^k$ leaves. Then $CSRS$ has:

$$\sum_{r=0}^{n} t_{\max}(n, r) = \frac{1}{3}n^2 + \frac{2}{3}.$$

*Proof.* The formula for $t_{\max}(n, r)$ for $CSRS$ is:

$$t_{\max}(n, r) = \begin{cases} 1 & \text{if } r = 0 \\ r(k - j) - 2(r - 2^j) & \text{if } 1 \leq r \leq n, \end{cases}$$

where $j = \lfloor \log_2(r) \rfloor$. We also know that $t_{\max}(n, n) = 0$, as this holds for all revocation schemes. So we can shorten the range of the sum as follows:

$$\sum_{r=0}^{n} t_{\max}(n, r) = 1 + \sum_{r=1}^{n-1} r(k - j) - 2(r - 2^j). \tag{C.1}$$

Since $j = \lfloor \log_2(r) \rfloor$, we have that $r = 2^j + a$ for some $0 \leq a \leq 2^j - 1$. So instead of summing over $r$, we can sum over both $j$ and $a$. This will make it easier to simplify the expressions:

$$\sum_{r=1}^{n-1} r(k - j) - 2(r - 2^j) = \sum_{j=0}^{k-1} \sum_{a=0}^{2^j-1} (2^j + a)(k - j) - 2(2^j + a - 2^j)$$

$$
\begin{aligned}
&= \sum_{j=0}^{k-1} \sum_{a=0}^{2^j-1} 2^j(k-j) + a(k-j) - 2(a) \\
&= \sum_{j=0}^{k-1} 2^j(2^j(k-j)) \sum_{a=0}^{2^j-1} a(k-j) - 2(a) \\
&= \sum_{j=0}^{k-1} 4^j(k-j) + \sum_{a=0}^{2^j-1} a(k-j-2) \\
&= \sum_{j=0}^{k-1} 4^j(k-j) + (k-j-2) \sum_{a=0}^{2^j-1} a \\
&= \sum_{j=0}^{k-1} 4^j(k-j) + (k-j-2)\frac{2^j(2^j-1)}{2} \\
&= \sum_{j=0}^{k-1} 4^j(k-j) + (k-j)\frac{4^j-2^j}{2} - 2\frac{4^j-2^j}{2} \\
&= \sum_{j=0}^{k-1} (k-j)\left(4^j + \frac{4^j-2^j}{2}\right) - (4^j-2^j) \\
&= \frac{3}{2}\sum_{j=0}^{k-1} 4^j(k-j) - \frac{1}{2}\sum_{j=0}^{k-1} 2^j(k-j) - \sum_{j=0}^{k-1} 4^j + \sum_{j=0}^{k-1} 2^j. \qquad \text{(C.2)}
\end{aligned}
$$

These summations evaluate to the following:

$$
\begin{aligned}
\sum_{j=0}^{k-1} 4^j(k-j) &= \frac{4}{9}4^k - \frac{1}{3}k - \frac{4}{9} \\
\sum_{j=0}^{k-1} 2^j(k-j) &= 2.2^k - k - 2 \\
\sum_{j=0}^{k-1} 4^j &= \frac{1}{3}4^k - \frac{1}{3} \\
\sum_{j=0}^{k-1} 2^j &= 2^k - 1.
\end{aligned}
$$

Substituting these back into Equation (C.2) gives:

$$
\begin{aligned}
\sum_{r=0}^{n} t_{\max}(n,r) =& 1 + \frac{3}{2}\left(\frac{4}{9}4^k - \frac{1}{3}k - \frac{4}{9}\right) - \frac{1}{2}(2.2^k - k - 2) \\
&- \left(\frac{1}{3}4^k - \frac{1}{3}\right) + 2^k - 1 = \frac{1}{3}4^k + \frac{2}{3} = \frac{1}{3}n^2 + \frac{2}{3}.
\end{aligned}
$$

□

# Bibliography


[1] Tomoyuki Asano. A revocation scheme with minimal storage at receivers. *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, 3108:433–450, 2002.

[2] Tomoyuki Asano. Reducing storage at receivers in sd and lsd broadcast encryption schemes. *Information Security Applications, 4th International Workshop, WISA*, pages 317–332, 2004.

[3] Tomoyuki Asano. Secure and insecure modifications of the subset difference broadcast encryption scheme. *Information Security and Privacy: 9th Australasian Conference, ACISP 2004*, pages 12–23, 2004.

[4] Nuttapong Attrapadung, Kazukuni Kobara, and Hideki Imai. Sequential key derviation patterns for broadcast encryption and key predistribution schemes. *Advances in Cryptology - ASIACRYPT 2003: 9th International Conference on the Theory and Application of Cryptology and Information Security*, pages 374–391, 2003.

[5] S. Berkovits. How to broadcast a secret. *Advances in Cryptology - CRYPTO '91, Lecture Notes in Computer Science*, 547:536–541, 1991.

[6] G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of the AFIPS 1979 National Computer Conference*, 48:313–317, 1979.

[7] R. Blom. An optimal class of symmetric key generation systems. *Proc. of the EUROCRYPT 84 workshop on Advances in cryptology: theory and application of cryptographic techniques*, pages 335–338, 1985.

[8] Weifeng Chen and Lakshminath R. Dondeti. Performance comparison of stateful and stateless group rekeing algorithms. *Proc. of Fourth International Workshop on Networked Group Communication, NGC*, 2002.

[9] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. *Lecture Notes in Computer Science*, 839:257–270, 1994.

[10] E. J. Harder D. M. Wallner and R. C. Agee. Key management for multicast: Issues and architectures. *IETF*, RFC 2627, 1997.

[11] Paolo D'Arco and Douglas R. Stinson. Fault tolerant and distributed broadcast encryption. *CT-RSA 2003, Lecture Notes in Computer Science*, 2612:263–280, 2003.

[12] Yevgeniy Dodis and Nelly Fazio. Public key broadcast encryption for stateless receivers. *ACM Workshop on Digital Rights Management, Lecture Notes in Computer Science*, 2696:61–80, 2002.

[13] Amos Fiat and Moni Naor. Broadcast encryption. *Advances in Cryptology - CRYPTO '93, Lecture Notes in Computer Science*, 773:480–491, 1994.

[14] Amos Fiat and Tamir Tassa. Dynamic traitor tracing. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 14(3):211–223, 2001.

[15] Juan A. Garay, Jessica Staddon, and Avishai Wool. Long-lived broadcast encryption. *Lecture Notes in Computer Science*, 1880:333–352, 2000.

[16] Dani Halevy and Adi Shamir. The LSD broadcast encryption scheme. *Advances in Cryptology - CRYPTO '02, Lecture Notes in Computer Science*, 2442:47–60, 2002.

[17] Kurosawa, Yoshida, Desmedt, and Burmester. Some bounds and a construction for secure broadcast encryption. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1998.

[18] Xiaozhou Steve Li, Yang Richard Yang, Mohamed G. Gouda, and Simon S. Lam. Batch rekeying for secure group communications. In *Proceedings of the tenth international World Wide Web conference on World Wide Web*, pages 525–534, Orlando, FL USA, 2001.

[19] Michael Luby and Jessica Staddon. Combinatorial bounds for broadcast encryption. *Advances in Cryptology - EUROCRYPT '98, Lecture Notes in Computer Science*, 1403:512–526, 1998.

[20] T. Matsumoto and H. Imai. On the key predistribution system: A practical solution to the key distribution problem. *Advances in Cryptology - CRYPTO '87, Lecture Notes in Computer Science*, 293:185–193, 1987.

[21] Miodrag J. Mihaljevic. Key management schemes for stateless receivers based on time varying heterogeneous logical key hierarchy. *Lecture Notes in Computer Science*, 2894:127–154, 2003.

[22] Dalit Naor and Moni Naor. Protecting cryptographic keys: The trace-and-revoke approach. *IEEE Computer*, 36:47–53, 2003.

[23] Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and tracing schemes for stateless receivers. *Advances in Cryptology - CRYPTO '01, Lecture Notes in Computer Science*, 2139:41–62, 2001.

[24] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. *Financial Cryptography: 4th International Conference, Lecture Notes in Computer Science*, 1962:1+, 2001.

[25] Carles Padro, Ignacio Gracia, Sebastia Martin, and Paz Morillo. Linear key predistribution schemes. *Designs, Codes and Cryptography*, 25:281–298, 2002.

[26] Carles Padro, Ignacio Gracia, Sebastia Martin, and Paz Morillo. Linear broadcast encryption schemes. *Discrete Applied Mathematics*, 128:223–238, 2003.

[27] A. Shamir R.L. Rivest and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, pages 120–126, 1978.

[28] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[29] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean. Self-healing key distribution with revocation. In Proceedings of IEEE Symposium on Security and Privacy:241–257, 2002.

[30] Chun Zhang Jim Kurose Weifeng Chen, Zihui Ge and Don Towsley. On dynamic subset difference revocation scheme. *Networking 2004, Lecture Notes in Computer Science*, 3042:743–758, 2004.

[31] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 68–79, 1998.

[32] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. Adding reliable and self-healing key distribution to the subset difference group rekeying method. *5th COST 264 International Workshop on Networked Group Communications, Lecture Notes in Computer Science*, 2816:107–118, 2003.