Carnegie Mellon University

Information Networking Institute

PROJECT

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Science in Information Networking

Title: **DYNAMIC SECRET REDISTRIBUTION**

Presented by: **SUJATA DOSHI**

Accepted by the Information Networking Institute

Thesis Advisor: DR. CHENXI WANG

_____

Signature **Date**

Thesis Reader: DR. DAWN SONG

_____

Signature **Date**

INI Director: DR. PRADEEP KHOSLA

_____

Signature **Date**

Project Presentation Date: 29th April 2004

TR# _____

Office use only

Acknowledgements

I would like to express my gratitude to my advisor Dr. Chenxi Wang who has provided invaluable guidance to me during my graduate studies at Carnegie Mellon University. We have worked very closely on this thesis and she has been very patient with me and has always encouraged me. I have really enjoyed my interactions with Chenxi and I consider it a great honor to have had the opportunity to work with her. I would like to thank my reader Dr. Dawn Song, for taking time to read my thesis and providing invaluable insights. I am grateful to my parents and my best friend Nikesh for their support and encouragement when I needed it the most.

Contents

Abstract

Verifiable secret sharing is a cryptographic primitive used in many distributed applications. To engineer realistic applications, it is beneficial to have dynamically changing shares and shareholder groups. Proactive secret sharing schemes deal with dynamically changing shares. Secret redistribution schemes solve issues of both dynamically changing shares and shareholder groups. In this dissertation, we identify the shortcomings of the traditional proactive secret sharing model and propose a new model, *Dynamic Secret Redistribution*, that allows for dynamically changing thresholds and group sizes. We describe this model for both synchronous and asynchronous systems and also develop dynamic secret redistribution schemes in accordance with these models.

Chapter 1

Introduction

1.1 Introduction To Secret Sharing

Secret Sharing is a technique of distributing trust among a group of participants. The primary motivation behind secret sharing is that a single node can be vulnerable to attack and compromise. Distributing the secret information to multiple nodes increases the confidentiality of the application employing the sharing scheme. More formally a secret s is divided into pieces called shares. These shares are distributed among nodes such that the pooled shares of specific subsets of nodes allows reconstruction of the original secret. A $(t+1, l)$ threshold secret sharing scheme for a secret s (distributed among l shareholders) is one such that $t+1$ or more shares pooled together can reveal s, but a group with less than $t+1$ shares cannot. In such a scheme an adversary must compromise $t+1$ or more shares to learn the secret and corrupt at least $l-t$ shares to destroy the information.

Secret sharing schemes were introduced independently by Blakley [Bla79] and by Shamir [Sha79] in 1979. Shamir used polynomial interpolation for secret sharing where the constant term in the polynomial corresponds to the secret. Blakley introduced a scheme where the intersection of $t+1$ of l vector spaces yields a one dimensional vector that represents the secret.

Verifiable Secret Sharing extends the original secret sharing schemes by tolerating a possibly corrupted dealer. Verification allows the shareholders to test the correctness of the shares they hold [Fel87, CGMA85, GMW87, Coh87, Ped92]. Verification is usually accomplished using witness

information distributed by the dealer. A witness value carries necessary information for the shareholders to verify their shares but does not reveal anything about the original secret.

The goal of secret sharing is greater security by distributing the knowledge of a secret and increasing the number of nodes an adversary must compromise to gain access to the secret. In the case of long-lived secrets, however, simple secret sharing is not sufficient—if the secret is distributed once to l nodes and never changed again throughout the lifetime of the secret, an adversary would have ample time to compromise enough shares and ultimately recover the secret. Proactive secret sharing [HJKY95, WWW02, CKLS02, Jar95, Rab98] overcomes this problem by dividing the life time of the secret into periodic refresh cycles. Within each cycle the shares of each node are refreshed without changing the original secret. Compromised old shares become useless once the shares are renewed, thereby resulting in greater security for long lived secrets. In order for the adversary to learn the secret, he must compromise at least threshold number of shares before the shares are refreshed. Proactive schemes make the assumption that within each refresh cycle less than threshold number of shares can be compromised by the adversary.

A class of proactive schemes also allows change in the threshold and the membership of the shareholder group. The notable ones include Desmedt and Jajodia's secret redistribution scheme [DJ97], an improved verifiable redistribution scheme introduced by Wong *et al.* [WWW02], and an RSA sharing scheme by Frankel *et al.* [FGMY97]. Desmedt and Jajodia described a non-interactive secret redistribution protocol for linear sharing schemes. In their protocol, each existing shareholder distributes share-of-shares to new shareholders, who then interpolate them to form new shares. Desmedt's scheme does not readily support verification. A malicious shareholder can pass fraudulent share-of-shares and result in faulty shares at the end of the redistribution. Wong *et al.* [WWW02] proposed a verification capability based on Desmedt and Jajodia's protocol. Frankel's scheme has a similar oversight that could allow malicious shareholders to corrupt the redistribution process. However the fix for Frankel's scheme differs from that of Desmedt's.

In this thesis we investigate dynamic groups and thresholds in the context of secret sharing schemes. We develop general redistribution models for both asynchronous and synchronous environments. We develop dynamic secret redistribution protocols in accordance with the proposed models. We

discuss the security of these protocols and also analyze their performance.

1.2 Motivation for Dynamic Secret Redistribution

Proactive secret sharing (PSS) provides protection to secrets through periodic updates of the shares. In a PSS scheme the lifetime of the secret is divided into epochs. At the beginning of each period the shareholders execute an update protocol, following which each shareholder has new shares of the original secret and all old shares are erased. A $(t+1, l)$ proactive scheme allows a maximum of t compromises and can tolerate up to $l - t - 1$ failures in each period.

Most proactive schemes, however, do not allow for dynamic groups and thresholds. The ability to change group size (and thus the members) and threshold provides control over the availability and security of the sharing scheme. For instance increasing the group size can potentially increase fault tolerance and availability of the secret. Dynamic group sizes also allow the ability to deal with changing requirements.

Dynamic thresholds provide the flexibility for changing the security of the system in accordance with a dynamic adversary whose abilities may change over time. Proactive update schemes assume a mobile adversary whose capabilities do not change with time. We call such an adversary a *rigid mobile adversary*. A rigid mobile adversary is one whose abilities are known before hand: the rate at which the adversary compromises servers is known. This knowledge makes it easy to model a proactive update system.

It is more difficult to model an adversary with changing abilities. We call such an adversary a *flexible mobile adversary*. A flexible mobile adversary is one whose abilities are not known before hand and can change with time. For instance such an adversary can increase or decrease the rate at which he compromises the servers based on the information he has learned over time. Such an adversary can become more powerful or less powerful with time.

Traditional proactive schemes do not deal with such an adversary. One possible way of dealing with a flexible mobile adversary is by adopting a reactive approach which is similar in all aspects

to the proactive approach except in the concept of static time intervals for updates. Instead of having scheduled proactive updates, updates are triggered when a threshold number of servers are compromised assuming there exists a mechanism which detects compromises automatically. In a $(t+1, l)$ system this threshold would lie between $[1, t]$. Thus in such a scheme the life time of a secret would be divided into updates that need not occur on a regular basis.

While the reactive approach can defend against a flexible mobile adversary, it does not allow for change in threshold and group size. In addition the updates are performed within the same group of servers. The possibility of replacing a server when it is compromised is not considered. These limitations motivate the need for a new way of secret sharing which we call dynamic secret redistribution.

1.3 Related work

As described previously, Shamir [Sha79] and Blakley [Bla79] were the first to introduce the concept of secret sharing. Desmedt introduced threshold cryptography that combines threshold secret sharing with cryptography [Des94]. In threshold cryptography, the cryptographic key is shared among a group of users. Each shareholder performs their share of the cryptographic operation independent of one another. The secret is revealed during the operation [Des97].

The notion of Verifiable Secret Sharing (VSS) allows shareholders to verify the authenticity of their shares in the event of a faulty dealer. Chor *et al.* introduced the notion of VSS in 1985 [CGMA85]. They present a scheme in which the dealer and the shareholders perform an interactive secure distributed computation for verification purposes. Feldman's VSS scheme [Fel87] is a non-interactive scheme in which a shareholder proves its shares validity independently of other shareholders in the group. Pedersen's scheme [Ped92] is also non-interactive like that of Feldman's. However Pedersen's scheme is more secure compared to Feldman's scheme. Benaloh [Coh87] describes a homomorphism property attained by several secret sharing schemes which allows multiple secrets to be combined by direct computation of shares. He also describes how his method can be used to simplify prior verifiable secret sharing schemes. Goldreich *et al.* [GMW87] present a zero-knowledge proof system that can be used to create a constant round interactive scheme for VSS for

a threshold t.

Ostrovsky and Yung introduce the concept of mobile adversaries [OY91] that corrupt participants in a distributed protocol and a constant rate. Jerecki *et al.* [Jar95, HJKY95] introduced the concept of Proactive Secret Sharing for Shamir's sharing scheme in which each shareholder periodically distributes update shares to all other shareholders.

Desmedt and Jajodia [DJ97] introduced the concept of share redistribution that allows shares to be refreshed periodically without changing the original secret. Once redistribution is completed, the old shares are erased. Any old shares that are compromised before redistribution are rendered useless. While Desmedt's redistribution scheme does allow for redistribution between different (possibly disjoint) sets of shareholders with different access structures, it does not support for verification–a faulty shareholder can therefore corrupt the redistribution process by sending spurious information during redistribution. Wong *et al.* identified the flaw in Desmedt's scheme and proposed a modification specialized for Shamir's sharing scheme [WWW02].

Frankel *et al.* proposed a proactive secret sharing scheme for RSA in [FGMY97]. Their scheme incorporates a combination of polynomial and additive sharing to achieve the refreshing of shares in each period. Frankel's scheme uses a poly-to-sum redistribution from polynomial sharing scheme to an additive sharing scheme. and a sum-to-poly redistribution from an additive sharing scheme to a polynomial sharing scheme. They claim that their scheme can incorporate dynamic group sizes and thresholds. However their verification relies on using public information distributed during the previous round to validate the correctness of the new shares. Their scheme will work if the same group of shareholders is used during share refreshing. However when new members join this group, they will be unable to perform these verifications because they do not have the corresponding public information. Wong *et al.* identify this flaw in Frankel's scheme and suggest a fix for it in [WWW02].

Rabin *et al.* [Rab98] also proposed a proactive RSA scheme. Their protocol is simpler than Frankel's protocol. They use an (l, l) additive sharing scheme to share the signature key. In order to provide for a threshold, they introduce the concept of a *share back up* where each share is

further shared using a (t, l) threshold scheme. Since their scheme uses additive sharing, refreshing of shares is fairly simple. However like Frankel's scheme their scheme also does not support for dynamic groups and thresholds.

Wong *et al.* [WWW02] introduced a protocol for Verifiable Secret Redistribution. Their protocol, unlike other proactive secret sharing protocols, can redistribute shares to arbitrary access structures. Their primary contribution is to allow the new shareholders to verify the validity of their shares after redistribution. Their scheme allows for dynamic changes in threshold a group size. Their scheme works only in the synchronous system environment.

Cachin *et al.* [CKLS02] and Zhou [Zho01] have independently proposed proactive secret sharing protocols for asynchronous systems. Zhou [Zho01] describes proactive secret sharing in the asynchronous environment using a concept of combinatorial secret sharing. Their scheme eliminates the assumption of a reliable broadcast channel which is generally assumed in all synchronous protocols for proactive secret sharing. Their scheme however is not very efficient and has a message complexity of $O(\binom{l}{t})$.

Cachin *et al.* [CKLS02] give general model and definition for proactive secret sharing in asynchronous systems. Their model is more formal and more efficient compared to Zhou's model. Cachin first describes a verifiable secret sharing protocol for asynchronous systems and uses this protocol as a building block in his proactive secret sharing model. Cachin's protocol also uses Threshold cryptographic primitives [CKS00] and a Multivalued Validated Byzantine Agreement protocol [CKPS01] and achieves a message complexity of $O(l^3)$. Since Cachin's proactive secret sharing protocol uses underlying threshold signature sharing schemes, it is difficult to efficiently extend Cachin's scheme to support for dynamically changing groups and thresholds.

1.4 Organization of this thesis

In Chapter 2 we describe some preliminaries of secret sharing. In Chapter 3 we describe a general model for secret redistribution in synchronous systems. We describe the general system model, the redistribution protocol, and present an analysis of the protocol. In Chapter 4 we describe an

implementation of secret redistribution protocol for synchronous systems and present our results. We move on to a general model for redistribution in asynchronous system in Chapter 5. We present preliminary building blocks needed for our asynchronous protocol followed by a general system model. We then explain our redistribution protocol and present an analysis of the same. In Chapter 6 we describe an implementation of the protocol for asynchronous systems and present our results. Finally we conclude this thesis in Chapter 7.

Chapter 2

Preliminaries of Secret Sharing

Let $\Gamma_{\mathcal{P}}^{(t+1,l)}$ denote the initial access structure where $\mathcal{P}$ denotes the initial set of shareholders, $|\mathcal{P}| = l$ and $t+1$ denotes the threshold of the sharing scheme. Let s denote the secret that has to be shared. In order to distribute s to the access structure $\Gamma_{P}^{(t+1,l)}$ threshold secret sharing schemes generally employ two functions:

1. A sharing function $f(\cdot,\cdot)$
2. A reconstruction function $r(\cdot)$.

For a $(t+1,l)$ sharing, the sharing function takes as input the secret s and generates an ordered l-tuple $(s_1,...s_l)$ employing the expression

$$s_i \leftarrow f(s,i) \tag{2.1}$$

The sharing function is a homomorphism from the secret space to the share space. Each $s_i, i = 1,...l$, is called a share of s and is distributed to each shareholder $i\epsilon\mathcal{P}$.

The reconstruction function takes as input the $t+1$-tuple $(s_1,...s_{t+1})$ and reconstructs the original secret s.

$$s \leftarrow r_{i\epsilon A}(s_i) \tag{2.2}$$

where $A, |A| \geq t+1$ denotes an authorized subset of the set of shareholders $\mathcal{P}$ belonging to access structure $\Gamma_P^{(t+1,l)}$. Our model assumes a particular class of sharing schemes called linear secret sharing. In such a scheme the secret can be written as

$$s = \sum_{i \epsilon A} r_i s_i \tag{2.3}$$

where r, the reconstruction function, is a homomorphism from the share space to the secret space.

To allow the shareholders to verify the correctness of their share we need to incorporate commitments (verification information) that are distributed along with the shares. We denote $E(\cdot)$ as the verification function. $E(\cdot)$ is also homomorphic that is

$$E(s_1 + s_2) = E(s_1) * E(s_2) \tag{2.4}$$

It is computationally infeasible to compute s_i from $E(s_i)$.

Applying the function E to Equation 2.1 and Equation 2.3 we get two verification conditions as follows:

$$E(s_i) \equiv E(f(s,i)) \tag{2.5}$$

$$E(s) \equiv E(\sum_{i \epsilon A} r_i s_i) = \prod_{i \epsilon A} E(r_i s_i) = \prod_{i \epsilon A} E(s_i)^{r_i} \tag{2.6}$$

Equation 2.5 allows a shareholder to ensure that his share s_i is equivalent to the result of the sharing function evaluated at i, that is $f(s,i)$. Equation 2.6 allows a shareholder to verify that the commitments $E(s_i), i \epsilon A$, correspond to the shares $s_i, i \epsilon A$, that can be used together to reconstruct to the correct secret s. Each shareholder can thus verify the correctness of his share s_i given the verification information $(E(f(s,i)), E(s), E(s_i)), i \epsilon A$

Chapter 3

Dynamic Secret Redistribution in Synchronous Systems

3.1 Model

In this section we propose a general system model for Dynamic Secret Redistribution from an access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to access structure $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$. The model is described for a synchronous system where all communication is in accordance with a common global clock. The model is reactive and tolerates a flexible mobile adversary.

System: The system initially comprises of a set of l servers $\mathcal{P}$ that share a secret s through a $(t+1, l)$ linear threshold scheme. The system must tolerate t faulty servers. In order to ensure that there exist at least $2t + 1$ correct servers in the system, we introduce the constraint $l \geq 3t + 1$.

Along with the share s_i each server $i\epsilon\mathcal{P}$ stores the verification information $E(s)$, which is the commitment to the original secret. The secret needs to be redistributed to a set of possibly disjoint l' servers $\mathcal{P}'$ such that $t' + 1$ or more correct servers can reconstruct the original secret. Note that $l' \geq 3t' + 1$.

All the servers are connected by a complete network of secure and authenticated point-to-point channels. They also have access to a common broadcast channel. The system is synchronized with respect to a common global clock. We assume that all honest servers will execute the protocol

operations and send the respective messages within a common time bound. We denote this bound as τ.

The Adversarial Model: Our model considers a flexible mobile adversary. The adversary can compromise less than threshold number of servers simultaneously.

Updates: To defend against a flexible mobile adversary dynamic updates are performed during the life of the secret. These updates are triggered when the number of servers compromised by the adversary reaches a threshold $\alpha, 1 \leq \alpha \leq t$. The updates can also be triggered when the requirements of the system have changed. The updates comprise of two sub-protocols; *secret redistribution protocol* which refreshes the old shares and *share recovery protocol* which recovers the corrupted new shares. We assume that the system has an built in detection mechanism to detect compromised servers.

Redistribution Period and Update phases: We define the period during a dynamic update as an *update phase* κ. In addition we define the period between update phase κ and update phase $\kappa + 1$ as a *redistribution period*.

3.2 Dynamic Update Protocol

In this section we describe our dynamic update protocol. As mentioned in the model, each update is invoked when the compromise threshold $\alpha, 1 \leq \alpha \leq t$ is reached or when the requirements of the system have changed. Each update invokes a secret redistribution protocol which redistributes shares from access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to access structure $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$. The redistribution protocol results in new shares held by each server belonging to $\mathcal{P}'$. These shares reconstruct to the same secret s. After the redistribution protocol some honest new shareholders may engage in a recovery protocol to recover their corrupted share of shares and consequently recover their new share.

3.2.1 Secret Redistribution Protocol:

In this section we present a definition for Dynamic Secret Redistribution in synchronous systems and present a protocol for redistribution from access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$. During redistribution

the new servers in $\mathcal{P}'$ engage in an agreement protocol to decide on a correct authorized set $A \subset \Gamma_{\mathcal{P}}^{(t+1,l)}$, which will be used for reconstruction of their new shares.

Definition 1 *A protocol for dynamic secret redistribution, in a synchronous system, from access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to ${\Gamma'}_{\mathcal{P}'}^{(t'+1,l')}$for a secret s, assuming a flexible mobile adversary who can compromise less than threshold number of servers in any redistribution period, satisfies the following conditions:*

Correctness: *The new shares of all the correct servers, at the end of each redistribution, can be used to reconstruct the secret s*

Secrecy: *An adversary, at any point of time during a redistribution period, who knows less than threshold number of shares and knows the verification information gains no information about the secret.*

Robustness: *The honest servers can reconstruct the correct secret s even in the presence of faulty shareholders. A robust redistribution protocol can tolerate t faulty old shareholders and t' faulty new shareholders provided there are at least $t+1$ correct old shareholders and at least $2t'+1$ correct new shareholders and $l \geq 3t+1$ and $l' \geq 3t'+1$.*

Flexibility: *The redistribution allows for*

- *Changing old threshold $t+1$ to a new threshold $t'+1$*
- *Changing old group size l to new group size l'.*

such that $l \geq 3t+1$ and $l' \geq 3t'+1$. This provides flexibility with respect to security and availability of the sharing scheme

The protocol is detailed in Figure 3.1. This protocol redistributes from access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to access structure ${\Gamma'}_{\mathcal{P}'}^{(t'+1,l')}$. We further impose the constraint that a maximum of t' new shareholders can be faulty and $l' \geq 3t'+1$ and a maximum of t old shareholders can be faulty and $l \geq 3t+1$. We denote faulty shareholders by over-bars.

In our protocol each new shareholder maintains a set $\mathcal{B}$ which is used locally to store the identities of the old shareholders who sent out incorrect commitment to the secret, that is $E(s)$. We denote

the faulty shareholders with over-bars. We assume that all $j \epsilon \mathcal{P}'$ have an enumerated list $List$ of authorized sets A chosen from $\mathcal{P} - \mathcal{B}$. This $List$ is the same for all correct $j \epsilon \mathcal{P}'$. During the protocol each new shareholder participates in an agreement protocol to decide on the authorized set A to use for reconstructing their new share. Each shareholder $j \epsilon \mathcal{P}'$ has a variable id, initialized to one, which identifies the current round of agreement. We remind the readers of our assumption that all correct shareholders will execute the protocol operations and send the necessary messages in bounded time. We denote this bound as τ. We assume there exists a global timer which is initialized to τ before each round of agreement commences.

We remind the reader that $f(\cdot, \cdot)$ is the sharing function, $r(\cdot)$ is the reconstruction function and $E(\cdot)$ is the verification function as described in Chapter 2. We assume initially each honest shareholder $i \epsilon \mathcal{P}$ has a valid share s_i and also has the verification information $E(s)$ which is the commitment to the original secret.

Referring to Step 1 in Figure 3.1, each shareholder $i \epsilon \mathcal{P}$ uses the sharing function to compute share of shares $s_{ij} \leftarrow f(s_i, j)$ and distributes these share of shares to all shareholders $j \epsilon \mathcal{P}'$ over the private channel. In Step 2 each shareholder uses the verification function to compute $E(f(s_i, j))$ and $E(s_i)$ and broadcasts $\{E(s), E(s_i), E(f(s_i, j))\}$.

In Step 3 each new shareholder first waits for $t + 1$ broadcasts with the same $E(s)$ to complete and then stores this as the correct value of $E(s)$. If a particular shareholder $\overline{i}$ has sent an incorrect commitment to the secret then $\overline{i}$ is added to the set of bad old shareholders $\mathcal{B}$. After all l broadcasts have completed each shareholder $j \epsilon \mathcal{P}'$ constructs the enumerated list $List$ of authorized sets $A, |A| = t + 1$ chosen from the set $(\mathcal{P} - \mathcal{B})$ in Step 5 of the protocol.

In order to ensure that the information sent by the shareholders $i \epsilon \mathcal{P}$ is correct each shareholder $j \epsilon \mathcal{P}'$ needs to perform certain verifications. In Step 6 $j \epsilon \mathcal{P}'$ finds the first authorized set A_{id} which satisfies Equation 2.6. This equation allows the shareholder $j \epsilon \mathcal{P}'$to verify that the commitments $E(s_i), i \epsilon A_{id}$ correspond to shares $s_i, i \epsilon A_{id}$ which when used together will reconstruct to the correct secret s.

To ensure that the commitment $E(s_i)$ corresponds to the share s_i used in the sharing function $f(s_i, j), i\epsilon\mathcal{P}, j\epsilon\mathcal{P}'$ we introduce one more verification condition. Since each honest shareholder $i\epsilon\mathcal{P}$ uses the sharing function $f(\cdot,\cdot)$ to distribute share of shares $s_{ij} = f(s_i, j)$, it follows that when we apply the reconstruction function $r(\cdot)$ to share of shares s_{ij} we will get s_i as follows.

$$s_i = \sum_{j\epsilon A'} r_j s_{ij} \tag{3.1}$$

Here $A' \subset \Gamma'^{(t'+1,l')}_{\mathcal{P}'}, |A'| \geq t'+1$. Applying the verification function $E(\cdot)$ to Equation 3.1 we get the following verification condition

$$E(s_i) \equiv E(\sum_{j\epsilon A'} r_j s_{ij}) = \prod_{j\epsilon A'} E(s_{ij})^{r_j} = \prod_{j\epsilon A'} E(f(s_i, j))^{r_j} \tag{3.2}$$

By performing this verification over the entire set $A' = \mathcal{P}'$, each shareholder $j\epsilon\mathcal{P}'$ can verify that $E(s_i)$ corresponds to the share s_i used in the sharing function $f(s_i, j), \forall j\epsilon\mathcal{P}', \forall i\epsilon A_{id}$.

Referring back to Step 6 of the protocol, each $j\epsilon\mathcal{P}'$ finds the first authorized set A_{id} in $List$ which satisfies Equation 2.6 and which satisfies Equation 3.2. In Step 7 j waits until the share of shares have been received from each $i\epsilon A_{id}$ and checks if the following equation holds

$$E(s_{ij}) \equiv E(f(s_i, j)) \tag{3.3}$$

for each $i\epsilon A_{id}$. This verification Equation 3.3 allows the shareholder $j\epsilon\mathcal{P}'$ to verify if his share of share s_{ij} is equivalent to the result of the sharing function evaluated at j and s_i, that is $f(s_i, j)$. In Step 7 if this verification checks out then $j\epsilon\mathcal{P}'$ broadcasts a commit message on this authorized set A_{id}, else he broadcasts an abort message on A_{id}.

Since we assume there can be a maximum of t' faulty new shareholders, and since we need to ensure that after redistribution there are at least $t'+1$ honest shareholders that have correct new shares, a maximum of $2t'+1$ commit messages are sufficient to arrive at an agreement on an authorized set A_{id}. $l'-2t'$ abort messages are sufficient to discard an authorized set A_{id}.

In Step 8 each new shareholder $j \epsilon \mathcal{P}'$ waits for the commit/abort messages to come in from each shareholder $k \epsilon \mathcal{P}'$ until the global timer set to τ has not expired. If there are $2t'+1$ or more commit messages on an authorized set A_{id} then j accepts the sharing and reconstructs his new share

$$s'_j = \sum_{i \epsilon A_{id}} r_i s_{ij} \tag{3.4}$$

If there $l' - 2t'$ or more abort messages or the timer has expired then $j \epsilon \mathcal{P}'$ sets $id \leftarrow id + 1$ and repeats the protocol from Step 6 on-wards. In the worst case it would take $\binom{l}{t+1} - \binom{l-t}{t+1}$ rounds, as derived in [WWW02], to arrive at an agreement on an authorized set A_{id}.

The redistribution scheme not only accomplishes share renewal but also allows for change in threshold $t+1$ to $t'+1$ and change in group size l to l' and $l \geq 3t+1$ and $l' \geq 3t'+1$. This enhances the traditional Proactive Secret Sharing scheme by providing flexibility of the sharing scheme.

3.2.2 Share Recovery Protocol:

In this section we describe our share recovery protocol which recovers the shares of honest new shareholders. It is possible that after redistribution some honest shareholders $j \epsilon \mathcal{W}, \mathcal{W} \subset \mathcal{P}'$ hold incorrect new shares. This situation can occur when the agreed upon authorized set A_{id} used to reconstruct new shares, contains some shareholder $\overline{k} \epsilon A_{id}$ that sent incorrect share of shares to shareholders $j \epsilon \mathcal{W}$. Note that in such a situation $\overline{k}$ must have broadcast correct verification information and must have sent correct share of shares to at least $t'+1$ honest shareholders $j \epsilon \mathcal{P}'$. To ensure that the availability of the system is intact, it is essential that the good shareholders engage in a recovery protocol which would recover their new share. The recovery protocol is detailed in Figure 3.2

Our recovery mechanism depends on the secret sharing homomorphism property introduced by Benaloh [Coh87]. Let $(s_1, s_2, \ldots s_l)$ denote shares of the secret s and $(s'_1, s'_2, \ldots s'_l)$ denote shares of the secret s'. A secret sharing scheme is called homomorphic if $(s_1 + s'_1, s_2 + s'_2, \ldots s_l + s'_l)$ is a possible share assignment of the secret $s + s'$. It can be easily deduced that linear secret sharing schemes are homomorphic.

After an agreement has been reached on the authorized set A_{id}, each good shareholder $j \epsilon \mathcal{P}'$ waits

till he receives the share of shares s_{kj} from each $k \epsilon A_{id}$. If s_{kj} does not satisfy the verification Equation 3.3 then k is added to set $\mathcal{R}$-set of identities of shareholders $\overline{k} \epsilon A_{id}$ that sent incorrect share of shares $\overline{s_{kj}}$ to shareholder j. j also determines a correct old shareholder $c \epsilon A_{id}$ that sent a correct share of share to j as indicated in Step 1 of the protocol. Since we assume that there can be a maximum of t faulty old shareholders, and since $|A_{id}| = t + 1$, we can be ensured that A_{id} will contain at least one correct old shareholder. In Step 2 j broadcasts a recovery request containing $\mathcal{R}$ and c.

In Step 3, each good shareholder $i \epsilon (\mathcal{P}' - \mathcal{W})$, ($\mathcal{W}$ being the set of shareholders that need recovery), on receiving a recovery request from shareholder $j \epsilon \mathcal{W}$ computes $\delta^i_{kc} = s_{ki} + s_{ci}$ for each $k \epsilon \mathcal{R}$. i then sends δ^i_{kc} to j over the private channel.

In order to allow shareholder j to ensure that δ^i_{kc} is correct we derive the following verification equation. We have,

$$s_{ci} \leftarrow f(s_c, i) \tag{3.5}$$

and

$$s_{ki} \leftarrow f(s_k, i) \tag{3.6}$$

Now applying $E(\cdot)$ to δ^i_{kc} we get

$$\begin{aligned} E(\delta^i_{kc}) &= E(s_{ki} + s_{ci}) \\ &= E(s_{ki})E(s_{ci}) \qquad (E(\cdot) \text{ IS HOMOMORPHIC}) \\ &= E(f(s_k, i))E(f(s_c, i)) \qquad (\text{APPLYING } E(\cdot) \text{ TO EQUATION 3.5 AND EQUATION 3.6}) \end{aligned}$$

In Step 4, j uses the equation

$$E(\delta^i_{kc}) \equiv E(f(s_k, i))E(f(s_c, i)) \tag{3.7}$$

to make sure that he has received the correct δ^i_{kc} from i.

In Step 5, after $j \epsilon \mathcal{W}$ receives recovery share of shares from $t'+1$ correct shareholders (say $\mathcal{C}$, denotes the set of these correct shareholders), j first recovers his share of share $s_{kj}, k \epsilon \mathcal{R}$ as follows

1. j first determines $s_k + s_c$,

$$\sum_{i \epsilon \mathcal{C}} r_i \delta_{kc}^i = \sum_{i \epsilon \mathcal{C}} r_i (s_{ki} + s_{ci}) \tag{3.8}$$

$$= \sum_{i \epsilon \mathcal{C}} r_i s_{ki} + \sum_{i \epsilon \mathcal{C}} r_i s_{ci} \tag{3.9}$$

$$= s_k + s_c \tag{3.10}$$

2. Then using the homomorphic secret sharing property of linear secret sharing, j determines $\delta_{kc}^j = s_{kj} + s_{cj}$

$$\delta_{kc}^j = f(s_k + s_c, j) \tag{3.11}$$

3. Finally j determines its correct share of share $s_{kj} = \delta_{kc}^j - s_{cj}$

After recovering all the incorrect share of shares j reconstructs its correct new share as $s'_j = \sum_{k \epsilon A_{id}} r_k s_{kj}$. This recovery protocol ensures that after redistribution all correct new shareholders will hold correct shares to the secret.

3.3 Analysis

In this section we present an analysis of the Secret Redistribution Protocol and the Share Recovery Protocol. We show that our redistribution protocol meets the properties of *Correctness, Secrecy, Robustness* and *Flexibility* defined in Definition 1. We also show that the recovery protocol meets properties of *Correctness, Secrecy* and *Robustness.*

Correctness:

Theorem 1 *For all secrets s, and all correct authorized sets $A \epsilon \Gamma_{\mathcal{P}}^{(t+1,l)}, |A| \geq t+1, |\mathcal{P}| = l$ and all correct authorized sets $A' \epsilon \Gamma'^{(t'+1,l')}_{\mathcal{P}'}, |A'| \geq t'+1, |\mathcal{P}'| = l'$, the new shares after redistribution reconstruct to the original secret s.*

Proof: The correctness proof is along the lines of the proof proposed by Wong *et al.* [WWW02]. Using Equation 2.3 we have

$$\begin{aligned} s &= \sum_{i \epsilon A} r_i s_i \qquad (\text{LINEAR SECRET SHARING}) \\ &= \sum_{i \epsilon A} \sum_{j \epsilon A'} r_i (r_j s_{ij}) \qquad (\text{LINEAR SHARE OF SHARES}) \\ &= \sum_{j \epsilon A'} \sum_{i \epsilon A} r_j r_i s_{ij} \qquad (a+b=b+a) \\ &= \sum_{j \epsilon A'} r_j \sum_{i \epsilon A} r_i s_{ij} \qquad (ab+ac=a(b+c)) \\ &= \sum_{j \epsilon A'} r_j s'_j \qquad (\text{EQUATION } 3.4) \end{aligned}$$

Theorem 2 *After recovery, the recovered shares of the correct servers along with the shares of the other correct servers, reconstruct to the original secret.*

Proof: Now the recovered shares will be correct provided the recovered share of shares are correct. We first prove that the recovered share of shares are correct. Consequently by the correctness property of redistribution, it follows that the recovered shares will be correct.

Let j denote the shareholder that needs recovery, let $\overline{s_{kj}}$ denote the corresponding share of share that needs recovery, let s_{cj} denote a correct share of share. To prove that the share of share s_{kj} is correctly recovered we have to prove that $s_{kj} = f(s_k, j)$.

Now through the recovery process j receives the recovery share of shares $\delta^i_{kc} = s_{ki} + s_{ci}$. When j receives $t'+1$ such recovery share of shares, j reconstructs $s_k + s_c = \sum_{i \epsilon C} r_i \delta^i_{kc}$ as explained before. Then j determines $\delta^j_{kc} = f(s_k + s_c, j)$. Due to the homomorphism property of secret sharing [Coh87] $\delta^j_{kc} = s_{kj} + s_{cj}$. Then j determines $s_{kj} = \delta^j_{kc} - s_{cj}$. Again due to the homomorphism property of secret sharing, since $s_{cj} = f(s_c, j)$ and $\delta^j_{kc} = f(s_k + s_c, j)$ it follows that $s_{kj} = f(s_k, j)$

Secrecy:

We need to show that an adversary at any point of time in the redistribution period who knows less than threshold number of shares and who knows the verification information will not be able to determine the secret.

Lemma 1 *An adversary who knows $\leq$ t shares of a redistribution period cannot determine the secret*

Proof: According to Equation 2.3, $t+1$ shares are necessary in order to reconstruct to the original secret. Now suppose that the adversary knows all shares of $i \epsilon A, |A| = t+1$, A being the authorized set chosen for reconstruction, except some share belonging to $i = k$. Equation 2.3 can be written as

$$s = r_k s_k + \sum_{i \epsilon A, i \neq k} r_i s_i \tag{3.12}$$

Since the adversary does not know s_k and s he cannot solve this equation because there are two unknowns and a single equation. Thus an adversary knowing only t shares will be unable to reconstruct the secret s. Correspondingly it follows that and adversary knowing less than t shares cannot reconstruct s.

Lemma 2 *An adversary who know $\leq$ t old shares and $\leq$ t' new shares of a given redistribution period will not be able to determine the secret s.*

Proof: We have $A \subset \Gamma_{\mathcal{P}}^{(t+1,l)}, |A| = t+1$ and $A' \subset \Gamma'^{(t'+1,l')}_{\mathcal{P}'}, |A'| = t'+1$. Let us assume that the adversary knows the shares $s_i, \forall i \epsilon A - \{k\}$ and shares $s'_j, \forall j \epsilon A' - \{p\}$. We can express the secret $s = \sum_{i \epsilon A} r_i s_i = \sum_{j \epsilon A'} r_j s'_j$ in terms of the following equations.

$$s = r_k s_k + \sum_{i \epsilon A, i \neq k} r_i s_i \tag{3.13}$$

$$s = r_p s'_p + \sum_{j \epsilon A', j \neq p} r_j s'_j \tag{3.14}$$

Let $\gamma = \sum_{i \epsilon A, i \neq k} r_i s_i$ and $\delta = \sum_{j \epsilon A', j \neq p} r_j s'_j$. Thus Equations 3.13 and 3.14 can be written as

$$s = r_k s_k + \gamma \tag{3.15}$$

$$s = r_p s'_p + \delta \tag{3.16}$$

From Lemma 1 we can deduce that $s \neq \gamma$ and $s \neq \delta$. We have to prove that an adversary knowing the terms γ and δ cannot determine the secret s. Let us assume that knowing γ and δ the adversary can determine s. In other words we have

$$s = r_k \delta + \gamma \tag{3.17}$$

Comparing Equation 3.15 and Equation 3.17 we have

$$s_k = \delta \tag{3.18}$$

From Equation 3.16 we have

$$\delta = s - r_p s'_p \tag{3.19}$$

From Equation 3.18 and Equation 3.19 we get

$$s_k = s - r_p s'_p \tag{3.20}$$

Substituting Equation 3.20 in Equation 3.15 we have

$$s = r_k(s - r_p s'_p) + \gamma$$

$$s(1 - r_k) = -r_k r_p s'_p + \gamma$$

$$s = -\frac{r_k r_p s'_p}{1 - r_k} + \frac{\gamma}{1 - r_k} \tag{3.21}$$

Comparing Equation 3.21 and Equation 3.15 we have

$$\frac{1}{1 - r_k} = 1 \tag{3.22}$$

$$s_k = \frac{r_p s'_p}{r_k - 1} \tag{3.23}$$

Equation 3.22 implies $r_k = 0$. Substituting $r_k = 0$ in Equation 3.21 we get $s = \gamma$ which is a contradiction. Also considering Equation 3.23 and Equation 3.18 we get

$$\delta = \frac{r_p s'_p}{r_k - 1} \tag{3.24}$$

Substituting Equation 3.24 in Equation 3.16 we get

$$s = r_p s'_p + \frac{r_p s'_p}{r_k - 1} \tag{3.25}$$

$$= r_p s'_p \frac{r_k}{r_k - 1} \tag{3.26}$$

Comparing Equation 3.25 and 3.16 we get $\delta = 0$ and $\frac{r_k}{r_k - 1} = 1$ which is a contradiction. Hence our original assumption was incorrect. It follows that an adversary knowing less than threshold number of shares for a given redistribution period cannot determine the secret s.

Theorem 3 *An adversary knowing less than threshold number of shares for any redistribution period cannot determine the secret s.*

Proof: We present a proof by induction. m identifies the redistribution period $m \epsilon \{1, \ldots n\}$. $\gamma = \sum_{i \epsilon A - k} r_i s_i$ as described in Lemma 2. Here A denotes the authorized set corresponding to the initial access structure. Let A'_m denote the authorized set corresponding to the access structure for redistribution period m. Also let $\delta_m = \sum_{j \epsilon A'_q, j \neq p_q} r_j s'_j$. We have to prove that an adversary knowing the terms γ and $\delta_m, \forall m \epsilon \{1 \ldots n\}$ cannot determine the secret s. In other words we need

to prove

$$s \neq \gamma + r_k \sum_{q=1}^{m} \delta_q \tag{3.27}$$

For $m = 1$ we have $s \neq \gamma + r_k \delta$ as proved in Lemma 2. Let us assume Equation 3.27 holds for $m = n - 1$. That is

$$s \neq \gamma + r_k \sum_{q=1}^{n-1} \delta_q \tag{3.28}$$

We have to prove Equation 3.27 holds for $m = n$. That is

$$s \neq \gamma + r_k \sum_{q=1}^{n} \delta_q \tag{3.29}$$

Let us assume

$$s = \gamma + r_k \sum_{q=1}^{n} \delta_q \tag{3.30}$$

Comparing Equation 3.30 to Equation 3.15 we have

$$s_k = \sum_{q=1}^{n} \delta_q \tag{3.31}$$

Now consider the system of equations

$$s = r_{p_1} s'_{p_1} + \delta_1 \tag{3.32}$$

$$s = r_{p_2} s'_{p_2} + \delta_2 \tag{3.33}$$

Similarly

$$s = r_{p_n} s'_{p_n} + \delta_n \tag{3.34}$$

From Equation 3.32,3.33,3.34 we get

$$\sum_{q=1}^{n} \delta_q = ns - \beta, \beta = \sum_{q=1}^{n} r_{p_q} s'_{p_q} \tag{3.35}$$

Substituting Equation 3.35 in Equation 3.31 we get

$$s_k = ns - \beta \tag{3.36}$$

Now substituting Equation 3.36 in Equation 3.15

$$s = r_k(ns - \beta) + \gamma$$

which implies

$$s = -\frac{r_k \beta}{1 - nr_k} + \frac{\gamma}{1 - nr_k} \tag{3.37}$$

Comparing Equation 3.15 and Equation 3.37 we get $1 - nr_k = 1$ which implies $r_k = 0$. Substituting $r_k = 0$ in Equation 3.37 we get $s = \gamma$. This is a contradiction. Hence our original assumption in Equation 3.30 was incorrect. Thus Equation 3.27 holds for $m = n$. Hence we can say an adversary knowing less than threshold number of shares for any redistribution period cannot determine the secret s.

Theorem 4 *During any point of time within a redistribution period an adversary who knows the verification information $\{E(s), E(f(s_i, j)), E(s_i)\}$ will not be able to learn any information regarding the secret.*

Proof: We remind the readers that the verification function $E(\cdot)$ is hard to invert. In other words it is computationally infeasible for the adversary to determine s from $E(s)$, $f(s_i, j)$ from $E(f(s_i, j))$ and s_i from $E(s_i)$. It follows that at any point of time within a redistribution period an adversary who knows the verification information will not be able to learn any information regarding the secret.

From Theorems 3 and 4 we have proved that the redistribution protocol preserves the secrecy of the secret from an adversary who either knows less than threshold number of shares or who knows the verification information for any redistribution period. However presently we are unable to provide a proof that the redistribution protocol preserves the secrecy of a secret from an adversary who knows less than threshold number of shares and knows the verification information for any redistribution period.

Theorem 5 *The recovery share of shares δ_{kc}^{i} do not reveal any information regarding the share s_i' of shareholder i.*

Proof: Without loss of generality, let us assume that the original access structure is $\Gamma_{\mathcal{P}}^{(2,l)}$, that is it has a threshold $t = 2$. In this case

$$\begin{aligned} s_i' &= r_k s_{ki} + r_c s_{ci} \\ &= r_k(s_{ki} + s_{ci}) + (r_c - r_k)s_{ci} \\ &= r_k \delta_{kc}^{i} + (r_c - r_k)s_{ci} \end{aligned}$$

Now even though δ_{kc}^{i} is known to the shareholder j requesting for recovery of s_{kj}, the shareholder does not know s_{ci}. In addition the probability that j can determine s_{ci} from δ_{kc}^{i} is negligible. Hence we can safely say that the recovery share of shares δ_{kc}^{i} does not reveal any information regarding the share s_i' of shareholder i.

Robustness:

Robustness implies correctness in the presence of faulty shareholders. The redistribution protocol can tolerate a maximum of t faulty old shareholders and t' faulty new shareholders. We denote the faulty shareholders and the information they send by over-bars.

Lemma 3 *Suppose $\overline{i}$ sends only incorrect share of shares $\overline{s_{ij}}$ to shareholder $j \epsilon \mathcal{P}'$ and broadcasts correct commitments, then a correct shareholder $j \epsilon \mathcal{P}'$ would broadcast $\{abort, A\}, \forall A \epsilon \mathcal{P}, \overline{i} \epsilon A$, A being the authorized set to be agreed upon for reconstruction.*

Proof: In this case while Equations 2.6 and 3.2 would hold, Equation 3.3 would not hold and consequently a correct new shareholder j would broadcast an abort message on the authorized set

A that contains $\overline{i}$.

Lemma 4 *Suppose $\overline{i}$ broadcasts incorrect commitment to the secret $\overline{E(s)}$, then $\overline{i}$ would be definitely present in the local bad set $\mathcal{B}$ of all correct shareholders $j\epsilon\mathcal{P}$ and $\overline{i}$ would not be included in any authorized set A to be agreed upon for reconstruction.*

Proof: We remind the readers of our assumption that there are at least $t+1$ old shareholders that are correct and a maximum of t are faulty. Thus in this case after receiving the same commitment value from $t+1$ or more old shareholders the new shareholder can arrive at the correct value of $E(s)$. Each new shareholder can thus identify the faulty old shareholder $\overline{i}$ based on inconsistent values of $E(s)$. Consequently each correct new shareholder j would add $\overline{i}$ to his local bad set $\mathcal{B}$ and would form a list $List$ of authorized sets A chosen from $(\mathcal{P}-\mathcal{B})$. As a result $\overline{i}$ would not be included in A.

Lemma 5 *Suppose $\overline{i}$ broadcasts an incorrect commitment to share $\overline{E(s_i)}$, then $A, \overline{i}\epsilon A$ would not be chosen for agreement by any correct $j\epsilon\mathcal{P}'$.*

Proof: In this case $A, \overline{i}\epsilon A$ when tested locally by each shareholder $j\epsilon\mathcal{P}'$ will not pass the verification Equation 2.6. Consequently a correct new shareholder $j\epsilon\mathcal{P}'$ would chose a different authorized set A that passes this verification equation. In this case however we cannot identify the faulty shareholder $\overline{i}$ unless we exhaustively perform the verification on all possible authorized subset $A\epsilon\mathcal{P}$.

Lemma 6 *Suppose $\overline{i}$ sends incorrect share of share $\overline{s_{ij}}$ and broadcasts an incorrect commitment $\overline{E(f(s_i,j))}$ which satisfies Equation 3.3, but $\overline{i}$ has used an incorrect share $\overline{s_i}$ when creating $\overline{s_{ij}}$, then $A, \overline{i}\epsilon A$ would not be chosen for agreement by any correct $j\epsilon\mathcal{P}'$.*

Proof: In this case $A, \overline{i}\epsilon A$ when tested locally by each shareholder $j\epsilon\mathcal{P}'$ will not pass the verification Equation 3.2 and Equation 3.3 simultaneously. Consequently a correct new shareholder $j\epsilon\mathcal{P}'$ would chose a different authorized set A that passes this verification equation.

Theorem 6 $2t'+1$ *commits on a set A are sufficient to arrive at a correct agreement among the new shareholders.*

Proof: We present a proof by contradiction. Let us suppose that there are $2t'+1$ or more commits but a correct agreement cannot been reached. This implies that even though we have $\geq 2t'+1$

commits, either the correctness of the redistribution has been violated or an agreement was not possible.

Consider the case that an agreement was not possible. This indicates that the new shareholders were unable to find a correct authorized set A which could be used for reconstructing new shares. This in turn indicates that there were $< t + 1$ correct old shareholders participating in the redistribution. But this is a contradiction since we assume that there will be at least $t + 1$ correct old shareholders during redistribution.

Now consider the case that an agreement on a set A has been reached but the correctness property is violated. This in turn indicates that the number of correct shareholders committing on A is less that $t' + 1$. This indicates that the number of faulty shareholders $> 2t' + 1 - t' - 1 = t'$. This is a contradiction since we assume a maximum of t' shareholders can be faulty. From both cases we can conclude that our original assumption was incorrect. Hence $2t' + 1$ commits on a set A are sufficient to arrive at a correct agreement.

Lemma 7 *Suppose a faulty new shareholder $\overline{j}$ commits on an authorized set A which does not satisfy the verification conditions then A may or may not be chosen for reconstruction*

Proof: In this case if a correct new shareholder $k\epsilon\mathcal{P}'$ gets $\geq 2t' + 1$ commits on A then he can accept A as proved in Theorem 6. If $k\epsilon\mathcal{P}'$ does not get as many commits on A, then all correct shareholders $k\epsilon\mathcal{P}'$ will not chose A for reconstruction.

Lemma 8 *The correct new shareholders will arrive at a correct agreement on some authorized set A even if a faulty new shareholder $\overline{j}$ broadcasts an abort on A when it actually passes the verification conditions*

Proof: In this case again if there are $2t' + 1$ commits the sharing can be accepted. Since there are at least $2t + 1$ correct old shareholders we are ensured that there is definitely one correct authorized set A. Further since $l' \geq 3t' + 1$, we know there are at least $2t' + 1$ correct new shareholders who would broadcast a commit on this correct authorized set A. Consequently it follows that all new shareholders would receive $2t' + 1$ commits on at least one authorized set A, even if all faulty new shareholders did not commit on any correct authorized set A. On receiving $2t' + 1$ commits, the correct new shareholders would reach an agreement on an authorized set A.

Theorem 7 *As long as there are $t' + 1$ correct new shareholders, the recovery protocol is robust.*

Proof: Robustness implies correctness in the presence of faulty shareholders. To provide for robustness, each shareholder that requests for recovery tests if the recovery share of share δ_{kc}^{i} satisfies Equation 3.7 before using it for recovery. In case δ_{kc}^{i} does not pass this check, it is not used in the recovery process. In addition since there are at least $t' + 1$ correct new shareholders present, it is guaranteed that the correctness of the recovery protocol will be preserved even in the presence of faulty shareholders.

Theorem 8 *As long as there are at least $t + 1$ correct old shareholders and at least $2t' + 1$ correct new shareholders, the correctness of the redistribution will be preserved even in the presence of faulty shareholders*

Proof: From Lemmas 3, 4, 5, 6 it can be seen that a sub-share s_{ij} sent by an old shareholder i will be chosen for reconstruction provided i has broadcast the correct commitments and s_{ij} passes Equations 3.3. It follows that a correct new shareholder j would not use $\overline{s_{ij}}$ for reconstructing his new share if $\overline{i}$ has not broadcast the correct commitments, or if $\overline{s_{ij}}$ does not pass the checks. From Lemmas 7 and 8 we can see that even in the presence of faulty new shareholders, the correct new shareholders will eventually arrive at a correct agreement on a correct authorized set A (an authorized A, such that $\forall i \epsilon A$, i has sent correct information to at least $t' + 1$ correct new shareholders). Hence we can state that a correct new shareholder j will always chose a correct authorized set A for reconstructing his new share. In addition since we have at least $2t + 1$ correct old shareholders, there would definitely be at least one correct authorized set A to be used for reconstruction of new shares. Also since an authorized set A is chosen for reconstruction, only if there are $2t' + 1$ commit messages on A, we can be ensured that after redistribution at least $t' + 1$ shareholders would hold valid shares. Consequently from Theorem 7 it follows that the correctness of the recovery protocol will be preserved. After recovery all honest shareholder $j \epsilon \mathcal{P}'$ have correct shares. It follows from the Theorem 1 that the correctness of the redistribution would be preserved. Thus we can say the redistribution protocol is robust.

Flexibility:

Theorem 9 *The redistribution protocol allows for flexible thresholds and group size.*

Proof: In the redistribution protocol the shares are redistributed from a $(t+1, l)$ threshold scheme to a $(t'+1, l')$ threshold scheme. Flexibility of threshold implies that the redistribution protocol allows for change from old threshold $t+1$ to new threshold $t'+1$. In other words after redistribution $t'+1$ new shareholders are necessary to reconstruct the secret. On similar lines flexibility of group size implies that redistribution allows for new shareholders to enter the group or old shareholders to leave the group. In other words after redistribution all l' new shareholders hold valid new shares to the original secret.

Now, the correctness proof demonstrates that after redistribution any authorized subset $A' \subset \Gamma'^{(t'+1,l')}_{\mathcal{P}'}$, $|A'| \geq t'+1, |\mathcal{P}'| = l'$ can reconstruct the original secret s. Since $|A'| \geq t'+1$ and $|\mathcal{P}'| = l'$, it shows that the protocol supported change in threshold from $t+1$ to $t'+1$ and change in group size from l to l'. Thus the redistribution protocol allows for flexible thresholds and flexible group size.

Dynamic Synchronous Secret Redistribution protocol for Linear Secret Sharing schemes

1. For each $i \epsilon \mathcal{P}$, use the sharing function $f(\cdot, \cdot)$ to compute share of shares $s_{ij} = f(s_i, j)$ and send s_{ij} to the corresponding $j \epsilon \mathcal{P}'$ over the private channel.
2. For each $i \epsilon \mathcal{P}$, use the verification function $E(\cdot)$ to compute $E(f(s_i, j))$ and $E(s_i)$ and send $\{E(s), E(s_i), E(f(s_i, j))\}$ to all $j \epsilon \mathcal{P}'$ over the broadcast channel.
3. For each $j \epsilon \mathcal{P}'$, wait till $\geq t + 1$ broadcasts containing identical values of $E(s)$ complete. Store this value of $E(s)$ as the correct value.
4. For each $j \epsilon \mathcal{P}'$, wait till all l broadcast messages come in. If there exists $\bar{i} \epsilon \mathcal{P}$ that sent an inconsistent value of $E(s)$ add $\bar{i}$ to $\mathcal{B}$.
5. For each $j \epsilon \mathcal{P}'$, form the list, $List$, of authorized subsets chosen from $\mathcal{P} - \mathcal{B}$.
6. For each $j \epsilon \mathcal{P}'$, find the first authorized set A_{id} in $List$ which has not been used for agreement before and which satisfies the following verification conditions.

$$E(s) \equiv \prod_{i \epsilon A_{id}} E(s_i)^{r_i}$$

$$E(s_i) \equiv \prod_{k \epsilon \mathcal{P}'} E(f(s_i, k))^{r_k}, \forall i \epsilon A_{id}$$

7. For each $j \epsilon \mathcal{P}'$, wait till share of shares s_{ij} are received from all $i \epsilon A_{id}$ and check if the following verification condition is satisfied.

$$E(s_{ij}) \equiv E(f(s_i, j)), \forall i \epsilon A_{id}$$

   If the verification condition checks out $\forall i \epsilon A_{id}$ then send a $\{commit, A_{id}\}$ message to all $k \epsilon \mathcal{P}'$ over the broadcast channel. In case the verification condition does not check out then broadcast an $\{abort, A_{id}\}$ message.
8. For each $j \epsilon \mathcal{P}'$, wait till the commit/abort messages come in from each $k \epsilon \mathcal{P}', k \neq j$ and the timer set to τ has not expired. If there are $\geq 2t' + 1$ commit messages on A_{id} accept the sharing and reconstruct new share s'_j.

$$s'_j = \sum_{i \epsilon A_{id}} r_i s_{ij}$$

   If there are $l' - 2t'$ or more abort messages or the timer has expired then set $id \leftarrow id + 1$, and repeat the protocol from Step 6 on wards.

Figure 3.1: Synchronous Secret Redistribution Protocol for redistribution of s from $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$

Share Recovery Protocol

1. For each $j \epsilon \mathcal{W}$, form a set $\mathcal{R}$ containing the identities of shareholders $\overline{k} \epsilon A_{id}$ that sent incorrect share of share $\overline{s_{kj}}$ to j. Also determine the shareholder $c \epsilon A_{id}$ that sent the correct share of share to j.
2. For each $j \epsilon \mathcal{W}$, broadcast a recovery request $\{\mathcal{R}, c\}$
3. For each $i \epsilon (\mathcal{P}' - \mathcal{W})$, on receiving a recovery request from shareholder j, compute $\delta_{kc}^{i} = s_{ki} + s_{ci}, \forall k \epsilon \mathcal{R}$, and send $\{\delta_{kc}^{i}\}$ to shareholder j over the private channel.
4. For each $j \epsilon \mathcal{W}$, on receiving the recover share of shares from shareholder i, verify that

$$E(\delta_{kc}^{i}) \equiv E(f(s_k, i))E(f(s_c, i))$$

5. For each $j \epsilon \mathcal{W}$, wait until receive recovery share of shares from $t' + 1$ shareholders that satisfy the above verification condition. (Let $\mathcal{C}$ denote this set of shareholders).
6. For each $j \epsilon \mathcal{W}$, recover share of share s_{kj} as follows
   (a) First compute $s_k + s_c = \sum_{i \epsilon \mathcal{C}} r_i \delta_{kc}^{i}$
   (b) Then compute $\delta_{kc}^{j} = f(s_k + s_c, j)$
   (c) Finally compute $s_{kj} = \delta_{kc}^{j} - s_{cj}$
7. For each $j \epsilon \mathcal{W}$, after recovering share of shares $s_{kj}, k \epsilon \mathcal{R}$, recover new share $s'_j = \sum_{k \epsilon A_{id}} r_k s_{kj}$.

Figure 3.2: Share Recovery Protocol

Chapter 4

Implementation of Dynamic Secret Redistribution in Synchronous Systems

4.1 Dynamic Secret Redistribution with Pedersen VSS

In this section we describe an implementation of a dynamic secret redistribution scheme based on the model proposed in the previous chapter. The basic cryptographic building blocks used in our redistribution protocol include the sharing function $f(\cdot,\cdot)$ which is implemented using Shamir's [Sha79] polynomial sharing scheme; the reconstruction function $r(\cdot)$ which is implemented using Lagrange's interpolation function; and the verification function $E(\cdot)$ which is implemented using Pedersen's verification scheme [Ped92].

4.1.1 Notation

Throughout the protocol p and q denote large primes such that q divides $p-1$. G_q is a unique subgroup of $\mathbb{Z}_p^*$ of order q, g is a generator of G_q, and h is an element of G_q such that nobody knows $log_g h$.

We use Shamir's sharing method where the sharing function f is a polynomial and the reconstruction function r is obtained through Lagrange's interpolation. The shareholders verify their shares using Pedersen's verification scheme.

In Pedersen's scheme the dealer commits himself to the secret $s \epsilon \mathbb{Z}_q$ by choosing $u \epsilon \mathbb{Z}_q$ at random and computing

$$E(s,u) = g^s h^u \mod p \tag{4.1}$$

where E denotes the verification function.

4.1.2 The Dynamic Update Protocol

We assume that initially the shareholders hold valid shares (s_i, u_i) to a secret (s,u) and also hold the commitment to the secret $E(s,u)$. The protocol comprises of a secret redistribution protocol followed by a share recovery protocol. As mentioned in our model, the redistribution from $(t+1,l)$ threshold scheme to $(t'+1,l')$ threshold scheme is initiated when the number of servers compromised by the adversary reaches the threshold α or when the requirements of the system have changed.

Secret Redistribution Protocol

This protocol redistributes from access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to access structure $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$, $l \geq 3t+1, l' \geq 3t'+1$. The protocol is detailed in Figure 4.1. The variables $\mathcal{B}$, $List$, id, $\tau$are as described in our model in Chapter 3. For the convenience of the reader we re-call what each variable denotes as follows

- $\mathcal{B}$ is a set of identities of faulty old shareholders $\overline{i} \epsilon \mathcal{P}$ that sent incorrect commitment to the secret $E(s,u)$.
- $List$ is an enumerated list of authorized sets A chosen from $(\mathcal{P} - \mathcal{B})$
- id identifies the current round of agreement on the authorized set A
- τ is the global time bound which signifies the time within which each correct shareholder must execute the protocol operations and send the necessary messages.

We assume that each shareholder has access to a broadcast communication channel and also that all shareholders are connected by means of point to point private channels. We assume that there

Dynamic Synchronous Secret Redistribution protocol for Shamir's Secret Sharing with Pedersen's Verification

1. For each $i \epsilon \mathcal{P}$, chose random coefficients $f_{ik} \epsilon \mathbb{Z}_q, k \epsilon \{1, \ldots, t'\}$ and use the polynomial $f'_i(j) = s_i + f'_{i1} j + \ldots + f'_{it'} j^{t'}$ to compute share of shares $s_{ij} = f'_i(j)$.
2. For each $i \epsilon \mathcal{P}$, chose random coefficients $v_{ik} \epsilon \mathbb{Z}_q, k \epsilon \{1, \ldots, t'\}$ and use the polynomial $v'_i(j) = u_i + v'_{i1} j + \ldots + v'_{it'} j^{t'}$ to compute $u_{ij} = v'_i(j)$
3. For each $i \epsilon \mathcal{P}$, send (s_{ij}, u_{ij}) to the corresponding $j \epsilon \mathcal{P}'$ over the private channel.
4. For each $i \epsilon \mathcal{P}$, compute $E'_{ik} = E(f'_{ik}, v'_{ik}), k \epsilon \{0, \ldots, t'\}$ and send $\{E(s,u), E'_{ik}\}$ to all $j \epsilon \mathcal{P}'$ over the broadcast channel. Note that $E(s_i, u_i) = E'_{i0}$
5. For each $j \epsilon \mathcal{P}'$, wait till $\geq t+1$ broadcasts containing identical values of $E(s,u)$ complete. Store this value of $E(s,u)$ as the correct value.
6. For each $j \epsilon \mathcal{P}'$, wait till all broadcast messages come in. If there exists $\overline{i} \epsilon \mathcal{P}$ that sent an inconsistent value of $E(s,u)$ add $\overline{i}$ to $\mathcal{B}$.
7. For each $j \epsilon \mathcal{P}'$, form the list, $List$, of authorized subsets chosen from $\mathcal{P} - \mathcal{B}$.
8. For each $j \epsilon \mathcal{P}'$, find the first authorized set A_{id} in $List$ which has not been used for agreement before and which satisfies the following verification condition.

$$E(s,u) = \prod_{i \epsilon A} E(s_i, u_i)^{r_i}, r_i = \prod_{m \epsilon A, m \neq i} \frac{m}{m-i} \tag{4.2}$$

   Note that r_i is determined using Lagrange's Interpolation function.
9. For each $j \epsilon \mathcal{P}'$, wait till share of shares (s_{ij}, u_{ij}) are received from all $i \epsilon A_{id}$ and check if $\forall i \epsilon A_{id}$ the following verification condition is satisfied.

$$E(s_{ij}, u_{ij}) = \prod_{k=0}^{t'} (E'_{ik})^{j^k}, i \epsilon A_{id} \tag{4.3}$$

   If the verification condition checks out $\forall i \epsilon A_{id}$ then send a $\{commit, A_{id}\}$ message to all $k \epsilon \mathcal{P}'$ over the broadcast channel. In case the verification condition does not check out then broadcast an $\{abort, A_{id}\}$ message.
10. For each $j \epsilon \mathcal{P}'$, wait till the commit/abort messages come in from each $k \epsilon \mathcal{P}', k \neq j$ and the timer set to τ has not expired. If there are $\geq 2t'+1$ commit messages on A_{id} accept the sharing and reconstruct new share $s'_j = \sum_{i \epsilon A_{id}} r_i s_{ij}$ and $u'_j = \sum_{i \epsilon A_{id}} r_i u_{ij}$ where r_i is determined using Lagrange's Interpolation function. If there are $\geq l' - 2t'$ abort messages or the timer has expired then set $id \leftarrow id + 1$ and repeat the protocol from Step 8 on wards. In the worst case it would take $\binom{l}{t+1} - \binom{l-t}{t+1}$ rounds, as derived in [WWW02], to reach an agreement on the authorized set A_{id}.

Figure 4.1: Synchronous Secret Redistribution Protocol for redistribution of s from $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$

are at most t faulty old shareholders, t' faulty new shareholders and $l \geq 3t+1$ and $l' \geq 3t'+1$. Redistribution proceeds as per the general protocol described in Chapter 3 and is summarized in Figure 4.1.

Share Recovery Protocol

Let $\mathcal{W} \subset \mathcal{P}'$ denote the set of shareholders that need recovery. The recovery protocol implementation for all $j \epsilon \mathcal{W}$ is detailed in Figure 4.2

Share Recovery Protocol

1. For each $j \epsilon \mathcal{W}$, form a set $\mathcal{R}$ containing the identities of shareholders $\overline{k} \epsilon A_{id}$ that sent incorrect share of share $\overline{s_{kj}}$ to j. Also determine the shareholder $c \epsilon A_{id}$ that sent the correct share of share to j.
2. For each $j \epsilon \mathcal{W}$, broadcast a recovery request $\{\mathcal{R}, c\}$
3. For each $i \epsilon (\mathcal{P}' - \mathcal{W})$, on receiving a recovery request from shareholder j, compute $\delta^i_{kc} = s_{ki} + s_{ci}$; $\gamma^i_{kc} = u_{ki} + u_{ci}, \forall k \epsilon \mathcal{R}$, and send $\{\delta^i_{kc}, \gamma^i_{kc}\}$ to shareholder j over the private channel.
4. For each $j \epsilon \mathcal{W}$, on receiving the recover share of shares from shareholder i, verify that

$$E(\delta^i_{kc}, \gamma^i_{kc}) \equiv \prod_{m=0}^{t'} (E'_{km})^{i^m} \prod_{m=0}^{t'} (E'_{cm})^{i^m} \tag{4.4}$$

5. For each $j \epsilon \mathcal{W}$, wait until receive recovery share of shares from $t'+1$ shareholders that satisfy the above verification condition. (Let $\mathcal{C}$ denote this set of shareholders).
6. For each $j \epsilon \mathcal{W}$, recover share of share s_{kj} as follows
   (a) First $\delta^j_{kc} = s_{kj} + s_{cj}$ by interpolating the recovery share of shares

$$\delta^j_{kc} = \sum_{i \epsilon \mathcal{C}} r_i \delta^i_{kc}, r_i = \prod_{m \epsilon \mathcal{C}, m \neq i} \frac{m-j}{m-i} \tag{4.5}$$

   (b) Then compute $s_{kj} = \delta^j_{kc} - s_{cj}$

   Similarly recover share of share u_{kj}.
7. For each $j \epsilon \mathcal{W}$, after recovering share of shares $s_{kj}, k \epsilon \mathcal{R}$, recover new share $s'_j = \sum_{k \epsilon A_{id}} r_k s_{kj}, u'_j = \sum_{k \epsilon A_{id}} r_k u_{kj}$ where r_k is determined using Lagrange's Interpolation function. .

Figure 4.2: Share Recovery Protocol

Complexity

In this section we evaluate the redistribution protocol on the basis of message complexity and computational complexity.

1. Message Complexity: During redistribution each shareholder $i \epsilon \mathcal{P}$ sends share of shares to each shareholder $j \epsilon \mathcal{P}'$. This results in ll' messages. Additionally there are l broadcast com-

mitment messages sent from the old shareholders to the new shareholders. In addition we have a maximum of l' commit and abort messages sent by the new shareholders in each round of agreement on the authorized set A. In the worst case we will have $(\binom{l}{t+1} - \binom{l-t}{t+1})$ rounds as in [WWW02]. During recovery in response to one recovery request broadcast message, we have a maximum $l' - 1$ point to point messages containing recovery share of shares.

2. Computational Complexity: The main factor to be considered here is the cost of computing the verifications Equation 4.2 , Equation 4.3 , Equation 4.4. Excluding the cost of computing the commitments, the verification conditions result in $O(tt')$ multiplications and $O(tt')$ exponentiations. In the worst case Equation 4.2 would have to be computed $\binom{l}{t+1} - \binom{l-t}{t+1}$ times, Equation 4.3 would have to be computed l times and Equation 4.4 would have to be computed $l' - 1$ times.

4.2 Results

We implemented the redistribution protocol using Shamir's secret sharing and Pedersen's verification as described in the previous section. We used the Open SSL Big Number Library for performing the modular arithmetic operations involved in the protocol. We allowed the user to vary the various redistribution parameters and evaluated our protocol based Computational Cost of verifications and End to End Cost of the protocol. The results of our tests are depicted in the following graphs.

In Figure 4.3 we plot Verification Cost against Share Size. We vary the share size from 16 bits to 1024 bits and plot the cost of computing the Verification Equations 4.2, 4.3 while redistributing from access structure $\Gamma_{\mathcal{P}}^{(2,4)}, |P| = 4$ to access structure $\Gamma'^{(2,4)}_{\mathcal{P}'}, |P'| = 4$. It is observed that the verification cost increases linearly with share size. This is expected because with increase in share size the cost of computing Equation 4.2 and Equation 4.3 will increase linearly, since the modular arithmetic operations have to be performed over larger numbers.

In Figure 4.4 we plot Verification Cost against Threshold. We vary the threshold from $t' + 1 = 2$ to $t' + 1 = 5$ and plot the cost of computing the Verification Equations 4.2, 4.3. The group size $l' = 13$ and the share size is kept constant at 512 bits. It is observed that as threshold increases the

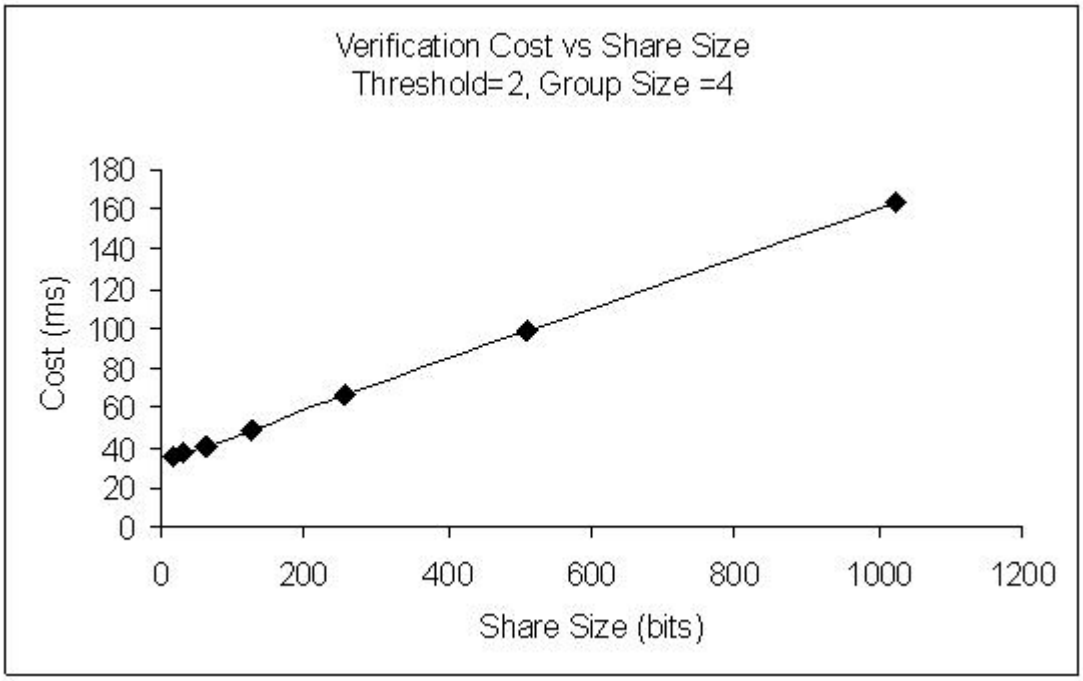

Figure 4.3: Verification Cost vs Share Size

cost of verification also increases linearly. This trend is expected because with increase in threshold, the number of exponentiations and multiplications that need to be performed in Equations 4.2,4.3 increases linearly.

In Figure 4.5 we plot Recovery Cost against Threshold. This is the cost of computing the Equation 4.4. In the experiment we kept one faulty old share holder $\bar{i}$ who sent an incorrect share of share $\overline{s_{ij}}$ to one correct new share holder j. The cost in the graph corresponds to the cost incurred by j when verifying the recovery share of shares corresponding to share of share $\overline{s_{ij}}$. We varied the threshold from $t' + 1 = 2$ to $t' + 1 = 5$, plot the cost of computing the Equation 4.4. The group size is kept constant at $l' = 13$ and the share size is kept constant at 512 bits. It is observed that as the threshold increases the recovery cost increases linearly. This trend is expected because with increase in threshold, the number of exponentiations and multiplications that need to be performed in Equation 4.4 increases linearly.

In Figure 4.6 we plot End to End Cost vs Share Size. End to End Cost is the time taken for the redistribution protocol to complete. This is measured starting from the point when the old shareholders

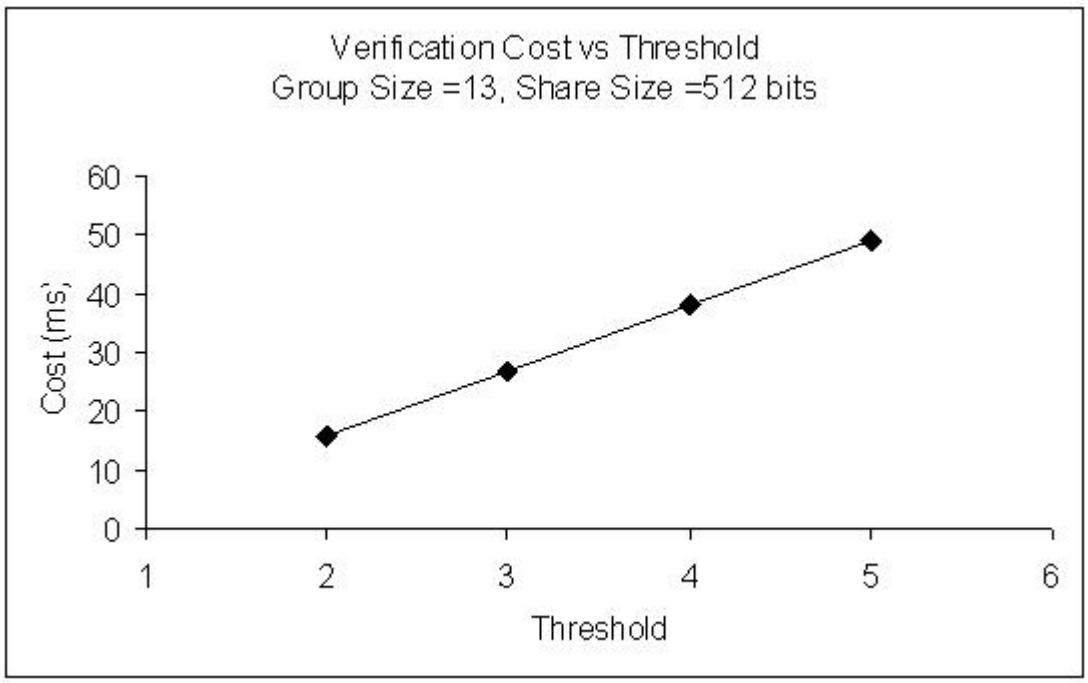

Figure 4.4: Verification Cost vs Threshold

compute share of shares to the point when the new shareholders reconstruct their new share. This includes Verification Cost and also includes network delay cost. In this graph we vary the share size from 16 bits to 512 bits and measure the End to End Cost of the protocol while redistributing from access structure $\Gamma_{\mathcal{P}}^{(3,7)}, |P| = 7$ to access structure $\Gamma'^{(3,7)}_{\mathcal{P}'}, |P'| = 7$. We plot the end to end cost for three case, In the first case labeled "No Restarts" agreement on a particular authorized set A was reached in the first round it self. In the second case labeled "1 Restarts" , it took two rounds of agreement to decide on an authorized set A. In the third case labeled "2 Restarts" it took three rounds of agreement to decide on an authorized set A. It is observed that End to End cost increases with share size. End to End cost comprises of verification cost of the Equations 4.2,4.3. Since verification cost increases with share size as depicted in Figure 4.3, it follows that end to end cost will increase with share size. Further the end to end cost also increases with the number of restarts. This is because as the number of restarts increase, the number of times Equation 4.2 has to be computed increases and the number of total messages increases. This increases verification cost and the total network delay cost, and consequently increases end to end cost.

In Figure 4.7 we plot End to End Cost vs Threshold. Group Size and Share Size are kept constant

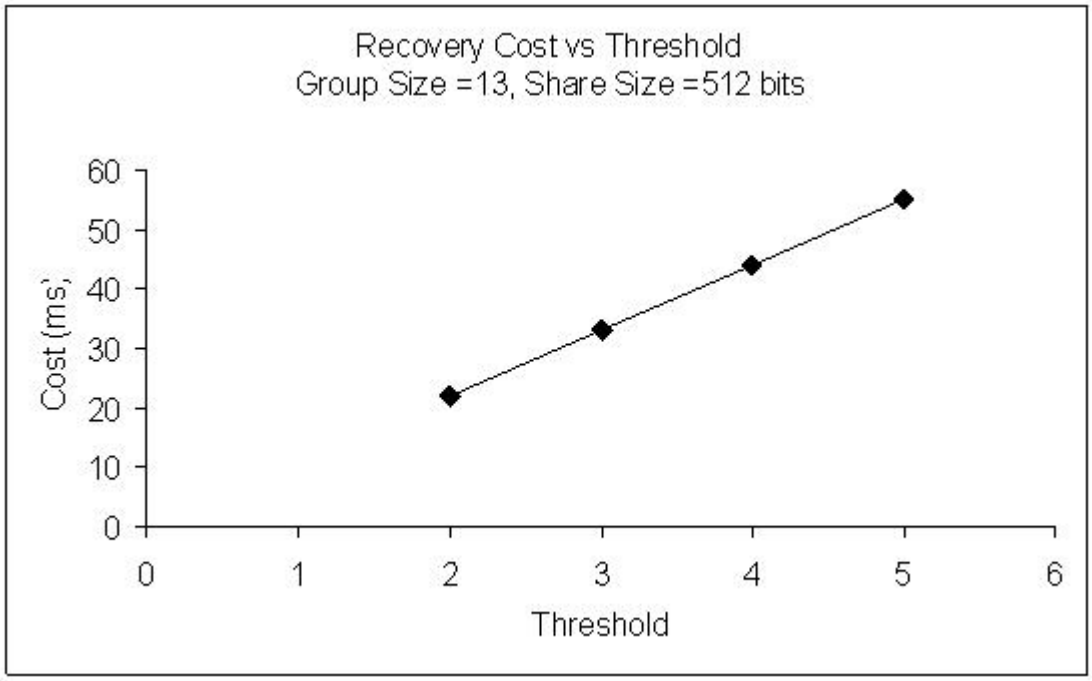

Figure 4.5: Recovery Cost vs Threshold

at $l' = 13$ and 512 bits respectively. We vary the threshold from $t' + 1 = 2$ to $t' + 1 = 5$ and determine the corresponding End to End Cost of the protocol. This cost includes the Verification Cost and also includes the network delay cost. It is observed that End to End Cost increases with increase in threshold. Since End to End Cost includes Verification Cost, and since Verification Cost increases with threshold as depicted in Figure 4.4, it follows that End to End Cost will increase with threshold.

In Figure 4.8 we plot End to End Cost vs Group Size. Threshold is kept constant at $t' + 1 = 2$ and Share Size is kept constant at 512 bits. We vary the group size from $l' = 4$ to $l' = 9$ and observe the corresponding End to End Cost. It is observed that the cost increases with increase in group size. This is expected because as group size increases the number of messages in the system increase, which increases the overall network delay cost, which further increases the End to End Cost.

In Figure 4.9 we plot End to End Cost vs Number of Recovery Requests. The threshold and group size are kept constant at $t' + 1 = 2$ and $l' = 7$ respectively. Share Size is kept constant at 512 bits. The End to End Cost plotted includes the Verification Cost, the Recovery Cost and the network delay

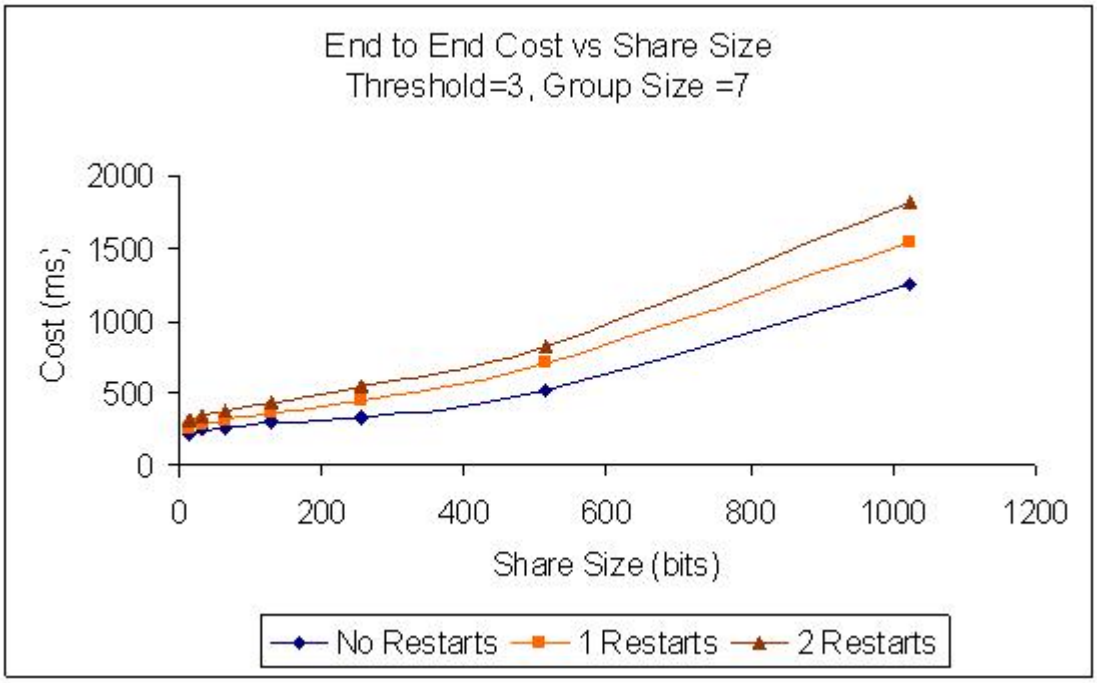

Figure 4.6: End to End Cost vs Share size

cost. It is observed that end to end cost increases linearly with the number of recovery requests. This is expected, since, as the number of recovery requests increase, the number of recovery messages in the system increases, which increases the network delay cost and consequently the end to end cost increases

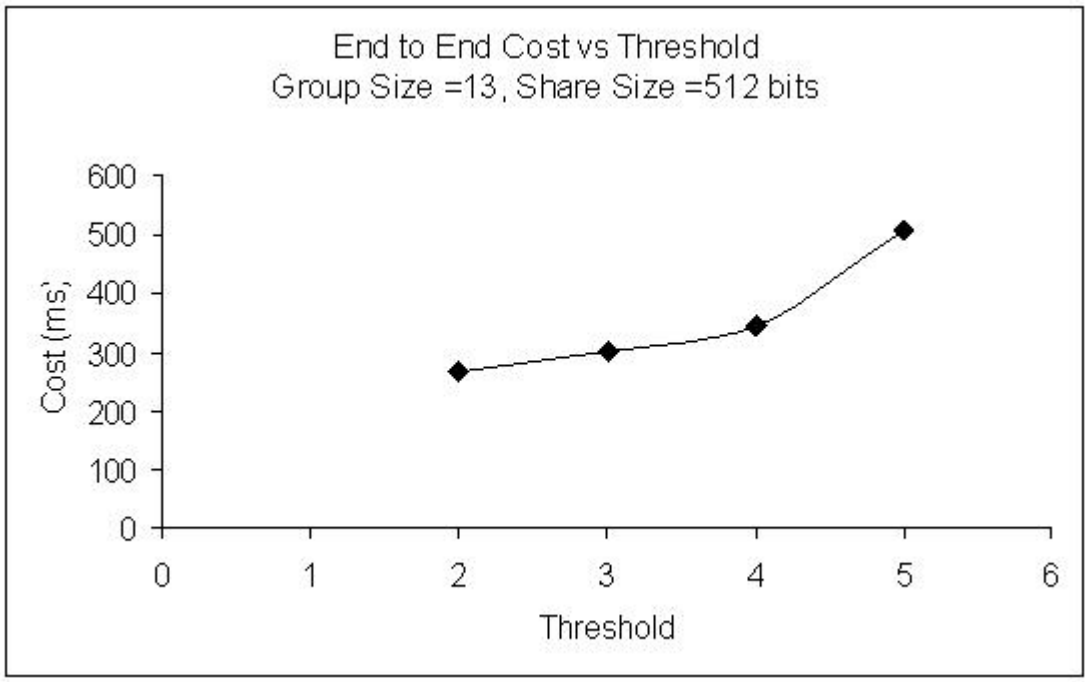

Figure 4.7: End to End Cost vs Threshold

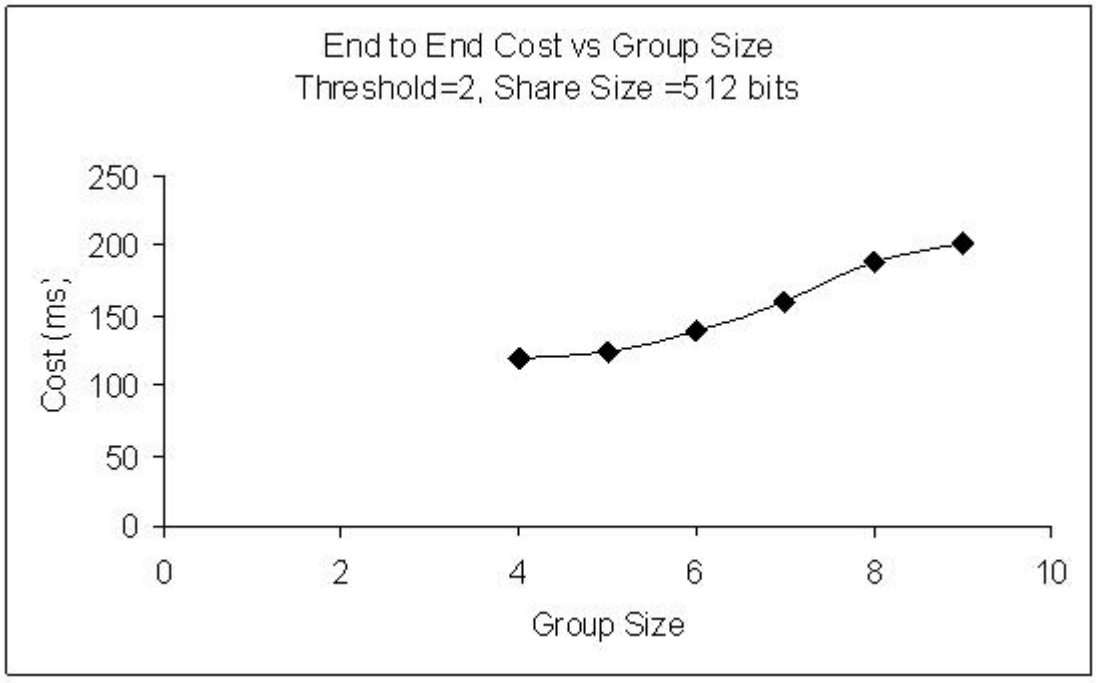

Figure 4.8: End to End Cost vs Group size

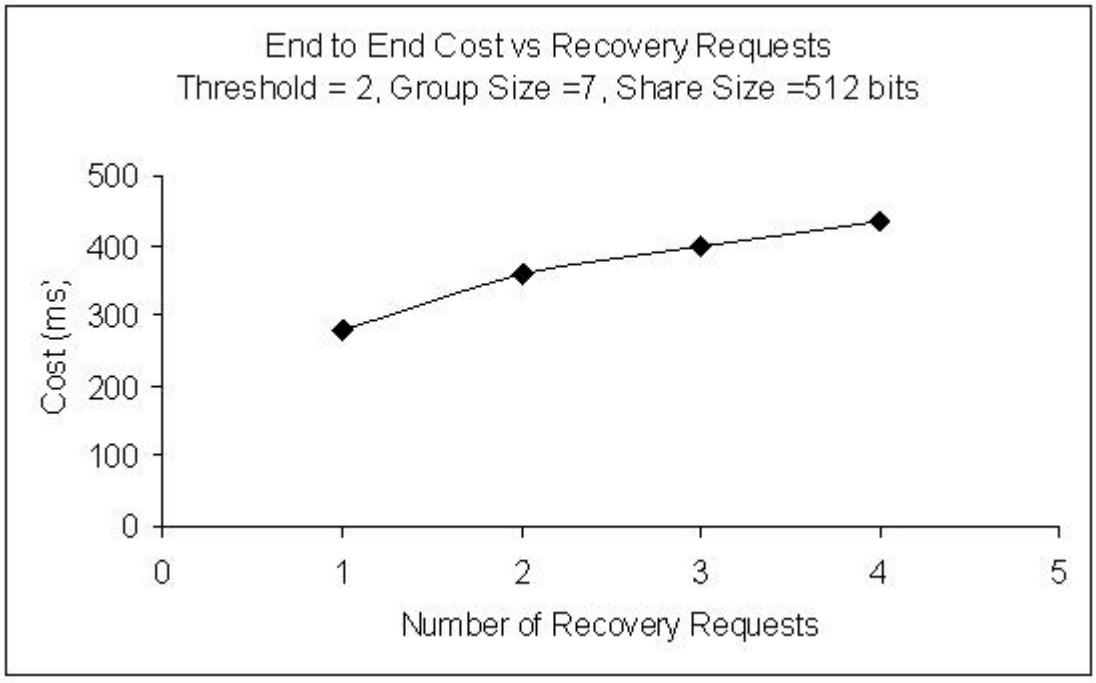

Figure 4.9: End to End Cost vs Number of Recovery Requests

Chapter 5

Dynamic Secret Redistribution Model for Asynchronous Systems

In this chapter we introduce a general model for Dynamic Secret Redistribution in Asynchronous Systems. We first compare our work with the proactive secret sharing model proposed by Cachin [CKLS02], and then move on to detail our model and protocol.

5.1 Related Work Comparison

In this section we compare our Dynamic Secret Redistribution model for Asynchronous Systems to that introduced by Cachin. Cachin *et al.* introduced a model and protocol for proactive secret sharing in [CKLS02]. Their asynchronous model assumes an adversary who can corrupt less than threshold $t + 1$ number of servers, where $t < \frac{l}{3}$, l being the group size. The adversary also controls the scheduling of messages. They first describe an Asynchronous Verifiable Secret Sharing (AVSS) protocol using Shamir's sharing scheme and Pedersen's Verification. Their scheme is implemented using bivariate polynomials, which allows for share of share recovery to be incorporated within the AVSS building block. They then propose a proactive secret refresh protocol using this AVSS building block and using a randomized multivalued byzantine agreement protocol [CKPS01]. Their refresh protocol has an expected message complexity of $O(l^3)$.

Cachin *et al.* protocol makes use of the threshold cryptographic primitives, Threshold Signature

Sharing and Threshold Coin Tossing. Each of these primitives uses a threshold sharing scheme. Since his protocol relies on such cryptographic primitives, it is difficult to extend his protocol efficiently to allow for flexibility in threshold and group size. This is because the threshold and group size would also have to be changed for the Threshold Signature Sharing and Threshold Coin Tossing scheme. If we eliminate these cryptographic primitives and try to extend Cachin's protocol for allowing flexible threshold and group size, then the message complexity increases to $O(l^4)$.

We introduce a model and protocol for Dynamic Secret Redistribution in Asynchronous Systems which is applicable to any linear threshold sharing scheme. In this model we assume a fair scheduler of messages. Our model also assumes an adversary who can corrupt less than threshold number of servers in a given redistribution period. Our protocol does not rely on threshold signature sharing or threshold coin tossing. Our protocol further allows for dynamically changing threshold and dynamically changing group size. The message complexity is the same as that of Cachin's, $O(l'^3), l'$ being the new group size. However in our protocol each server has to wait for more number of messages to come in, compared to Cachin's protocol, before certain protocol operations are executed - that is the wait time in our protocol is higher compared to Cachin's protocol. Since we do not use Threshold Signature Sharing , Threshold Coin Tossing, the computational cost of verification of signature shares is not present in our protocol.

5.2 Model and Assumptions

In this section we propose a general system model for redistribution in an asynchronous environment. In asynchronous networks there is no common clock and the messages can be arbitrarily delayed. This makes it difficult to define a proactive model as the servers have no common notion of time. Our asynchronous protocol build on top of reliable broadcast and asynchronous consensus protocols. We first briefly explain each of these building blocks and then describe our Asynchronous Secret Redistribution protocol. We remind the readers that l denotes the initial group size and $t+1$ denotes the initial threshold of the sharing scheme and $l \geq 3t+1$. After redistribution l' servers share the secret using a $(t'+1, l')$ sharing scheme and $l' \geq 3t'+1$

5.2.1 Preliminary Building Blocks

In this section we describe preliminary building blocks used in our Secret Redistribution protocol.

Echo Broadcast

Here we briefly describe the Echo Broadcast protocol proposed by Bracha [BT83]. The basic messages in the protocol are $initial, echo$ messages. The transmitter sends an $(initial, v)$ message to all processes. On receiving this message the processes report to each other the value v via $(echo, v)$ messages. If a process receives more than $\frac{(l+t)}{2}$ echo messages on the same value v, then it $Accept$ v. The above protocol has a message complexity of $O(l^2)$

$EchoBroadcast(v, t, l)$

1. $Send(initial, v)$ message to all the processes. (Only by the transmitter)
2. Upon $Receive$ for some v an $(initial, v)$ message for the first time, $Send$ $(echo, v)$ message to all processes.
3. Upon $Receive$ for some v $\frac{l+t}{2}$ echo messages $Accept$ v.

Figure 5.1: An Echo Broadcast Protocol for $t < \frac{l}{3}$

Reliable Broadcast

We briefly describe the Reliable Broadcast protocol proposed by Bracha [Bra84]. The basic messages in the protocol are $initial, echo$ and $ready$ messages. The transmitter sends an $(initial, v)$ message where v denotes the value on which an agreement has to be made. The processes then report to each other via $(echo, v)$ messages. If a process receives more than $\frac{(l+t)}{2}$ $(echo, v)$ messages then it sends a $(ready, v)$ message to all other processes. Also if a process receives $t+1$ $(ready, v)$ messages then it sends its own $(ready, v)$ message. On receiving $2t+1$ $(ready, v)$ messages with the same value v, the process accepts v. The protocol is detailed in Figure 5.2. The protocol has a message complexity of $O(l^2)$

Binary Asynchronous Consensus Protocol

Here we describe the binary consensus protocol proposed by Bracha and Toueg in [BT85]. This is used when all the new shareholders want to come to a consensus regarding the authorized set to

$ReliableBroadcast(v, t, l)$

1. $Send(initial, v)$ message to all the processes. (Only by transmitter)
2. Wait till $Receive$ for some v, one $(initial, v)$ message, or $\frac{(l+t)}{2}$ $(echo, v)$ messages, or $(t+1)$ $(ready, v)$ messages. $Send(echo, v)$ message to all the processes.
3. Wait till it $Receive$ for some v, $\frac{(l+t)}{2}$ $(echo, v)$ messages or $(t+1)$ $(ready, v)$ messages. $Send(ready, v)$ message to all the processes.
4. Waits till it $Receive$ for some v, $2t+1$ $(ready, v)$ messages. $Accept$ v.

Figure 5.2: A Reliable Broadcast Protocol for $t < \frac{l}{3}$

use for reconstructing the new shares. The protocol makes use of the $EchoBroadcast$ primitive described earlier. Initially each process sends its proposed value to all other processes using the $EchoBroadcast$ primitive. Note that each echo broadcast is tagged with the corresponding round identification rid. Each process waits till it accepts proposals from $l-t$ processes. Then it changes its value to the majority of the values of the accepted messages. If a process receives more than $\frac{l+t}{2}$ messages with the same value v then the process decides on v. When a process p decides on a value v it sends to all the processes the message $(initial, v, *)$ and echoes of the form $(echo, v, *)$ for all other processes q. On receiving these terminating messages a process sends them to itself. The above protocol has a message complexity of $O(l^3)$ for one round of the protocol. The expected

AsynchronousConsensus(t,l)

1. Initialize rid to 0
2. $EchoBroadcast$ the proposal value v_{rid} for the current round
3. Wait till $Accept$ $l-t$ proposals and then set the proposal value for the next round v_{rid+1} as the majority of the values accepted in round rid
4. If $Accept$ $\frac{l+t}{2}$ proposals with the same value v_{rid} then decide on v and send $(initial, v, *)$ and $(echo, v, *)$ to all processes and exit from loop
5. Increment rid and repeat protocol from Step 2 on wards

Figure 5.3: A Binary Asynchronous Consensus Protocol for $t < \frac{l}{3}$ for Process P

number of rounds to convergence is a constant if $t = O(\sqrt{l})$. If $t = O(l)$ then the number of rounds to convergence is exponential.

Multivalued Asynchronous Consensus Protocol

Here we describe the multivalued consensus protocol introduced by [Tou84]. The protocol reduces the multivalued agreement to a binary consensus protocol. The result of this protocol will be one

of the the proposed values provided enough correct processes have proposed the same value. If no such value exists then the processes agree on the value 0. The protocol uses Digital Signatures. We assume that these keys used for signing are stored securely or are proactively refreshed. The protocol is described in Figure 5.4. Since the protocol executes a single binary consensus protocol,

$MultivaluedConsensus(t, l)$ (Phase 1)

1. $Send$ the initial proposal v to all the processes
2. Upon $Receive$ proposals from $l - t$ distinct processes
   (a) Set Π as the set of signed received messages
   (b) Set m as the value that occurs the most in all the proposals
   (c) Set $count(m)$ as the number of proposal messages on m
   (d) if $count(m) \geq l - 2t$ then set $v = m$ else set $v = 0$

(Phase 2)

1. $EchoBroadcast$ v and proof Π.
2. Wait till $Accept$ the proposals with a correct proof from $l - t$ distinct processes.
   (a) Set m as the value that occurs most among all the accepted proposals.
   (b) Set $count(m)$ as the number of accepted proposals on m
   (c) If $count(m) = l - t$ then set $v = m$ else set $v = 0$.

If $count(m) > \frac{l-t}{2}$ then propose m in a Binary Asynchronous Consensus protocol which performs consensus on proposals in the set $\{m, 0\}$. If $count(m) \leq \frac{l-t}{2}$ then perform a consensus on $\{0\}$.

Figure 5.4: A Multivalued Asynchronous Consensus Protocol for $t < \frac{l}{3}$ for Process P

the message complexity of the protocol is $O(l^3)$.

5.2.2 Asynchronous System Model

In this section we describe our asynchronous model which basically includes an asynchronous network of servers with a computationally bounded adversary.

System. The system initially consists of a set $\mathcal{P}$ of l servers that share a secret s through a $(t + 1, l), l \geq 3t + 1$ threshold secret sharing scheme. The secret has to be redistributed to a possibly disjoint set $\mathcal{P}'$ of l' servers such that after redistribution these servers will share the secret s through a $(t'+1, l'), l' \geq 3t'+1$ threshold secret sharing scheme. We assume every pair of servers is linked by a *secure asynchronous channel* that provides privacy and authenticity. The system is asynchronous.

Communication Primitives. There are three basic communication primitives $Send$, $Receive$ and $ReliableBroadcast$. The $Send$ and $Receive$ primitives are used to send and receive messages on the secure asynchronous channel. The $ReliableBroadcast$ primitive described in the previous section allows for reliable broadcast of messages in the network.

Adversary. We assume a flexible mobile adversary whose capabilities change with time. The adversary can corrupt less than threshold number of servers simultaneously.

Scheduler. We assume a fair scheduler of messages. We define $R(p, q, k)$ to be the event that p receives a message from q in round k. A scheduler is said to be fair provided the following conditions hold [BT85]

1. For any process p and q, and round k there is a positive constant ϵ such that $Pr[R(p, q, k)] > \epsilon$
2. For any distinct processes r, p and q in round k the events $R(q, r, k)$ and $R(q, p, k)$ are independent.

Updates: To defend against a flexible mobile adversary dynamic updates are performed during the life of the secret. These updates are triggered when the number of servers compromised by the adversary reaches a threshold $\alpha, 1 \leq \alpha \leq t$. The updates can also be triggered when the requirements of the system have changed. The updates comprise of two sub-protocols; secret redistribution protocol which refreshes the old shares and share recovery protocol which recovers the corrupted new shares. We assume that the system has an inbuilt detection mechanism to detect compromised servers.

Redistribution Period and Update phases: We define the period during the dynamic update as an *update phase* κ. In addition we define the period between update phase κ and update phase $\kappa + 1$ as a *redistribution period*.

Messages. Our messages are modeled along the lines of the model proposed by Cachin *et al.* [CKLS02]. There exists a global set of messages $\mathcal{M}$, whose elements are tagged by a label $a : b : \beta$ denoting the sender a, receiver b , and identifier β for the current redistribution period. If server j enters the redistribution period β then all messages in $\mathcal{M}$ with labels $j : \cdot : \sigma$ where $\sigma < \beta$ are removed from $\mathcal{M}$. Further the scheduler may not schedule messages with label $j : \cdot : \beta$ before j

has entered the redistribution period β. If an adversary corrupts a server j during run β, then all messages with labels $j:\cdot:\beta$ are given to the adversary and now the adversary may send messages with label $j:\cdot:\beta$.

5.3 Dynamic Update Protocol

In this section we describe a dynamic update protocol which consists of the secret redistribution protocol and a share recovery protocol. This update protocol is invoked each time the requirements of the system have changed or the number of compromises reaches the threshold α. We assume the shareholders initially start with a valid sharing of secret s. Additionally the shareholders also have the commitments $E(f(s,i)), \forall i \epsilon \mathcal{P}$ distributed during the initial share distribution.

5.3.1 Secret Redistribution Protocol

In this section we describe a secret redistribution protocol which meets the following definition

Definition 2 *A protocol for dynamic secret redistribution, in an asynchronous system, from access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$ for a secret s, assuming an adversary that corrupts less than threshold number of servers in a given redistribution period, satisfies the following conditions:*

Liveness:*If the scheduler activates all honest shareholders and delivers all associated messages within a redistribution period, then all honest shareholders complete the redistribution except with negligible probability.*

Correctness: *The new shares,of the correct shareholders, at the end of each redistribution can be used to reconstruct the secret s*

Secrecy: *An adversary, at any point of time during a redistribution period, who knows less than threshold number of shares and who knows the verification information gains no information about the secret.*

Robustness: *The protocol is said to be robust if the honest servers can reconstruct the correct secret s even in the presence of faulty shareholders. A robust redistribution protocol can tolerate t faulty old shareholders and t' faulty new shareholders provided there are at least $t+1$ correct old shareholders and at least $2t'+1$ correct new shareholders and $l \geq 3t+1$ and $l' \geq 3t'+1$.*

Flexibility: *The redistribution allows for*

- *Changing old threshold $t+1$ to a new threshold $t'+1$*
- *Changing old group size l to new group size l'.*

such that $l \geq 3t+1$ and $l' \geq 3t'+1$. This provides flexibility with respect to security and availability of the sharing scheme

This protocol redistributes from access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to access structure $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$. Each shareholder $j\epsilon\mathcal{P}'$ initializes a local variable $count\_phase = 0$. We assume that all $j\epsilon\mathcal{P}'$ have an enumerated list $List$ of authorized sets chosen from $\mathcal{P}$. This $List$ is the same for all correct $j\epsilon\mathcal{P}'$. Each shareholder has two local sets X and $\mathcal{I}$. X is used to store the identities of the old shareholders that pass the verification equations. $\mathcal{I}$ is used to store the identities of those old shareholders whose messages have been received. In addition we assume each new shareholder is equipped with a local timer which is set to a specific time out $\mathcal{T}$. We assume that the scheduler schedules messages such that all the messages from the honest servers arrive before the timer expires. In case such scheduling is not possible, then the new shareholders just have to wait till the messages come in from the honest old shareholders. We remind the readers that $f(\cdot,\cdot)$ denotes the sharing function, $r(\cdot)$ denotes the reconstruction function, and $E(\cdot)$ denotes the verification function. The redistribution protocol is detailed in Figure 5.5. Note that Step 7 to Step 10, denotes a phase of the protocol and Step 5 to Step 11 denotes a round of the protocol. In a maximum of two rounds the protocol will complete. Further each round will contain a maximum of two phases. $count\_phase$ keeps track of the current phase of the protocol.

In Step 1 of the redistribution protocol in Figure 5.5 each old shareholders $i\epsilon\mathcal{P}$ creates a share of share s_{ij} and distributes this share of share privately to each new shareholder $j\epsilon\mathcal{P}'$. In Step 2 each old shareholder i reliably broadcasts the verification information $E(s_i), E(f(s,k)), E(f(s_i,j))$, $\forall k\epsilon\mathcal{P}$, $\forall j\epsilon\mathcal{P}'$.

In Step 3, each shareholder $j\epsilon\mathcal{P}'$ waits for the scheduler to deliver $t+1$ reliable broadcasts containing identical values of $E(f(s,i)), i\epsilon\mathcal{P}$. This value is stored as the correct value of $E(f(s,i)), \forall i\epsilon\mathcal{P}$.

Dynamic Asynchronous Secret Redistribution protocol for Linear Secret Sharing schemes

1. For each $i \epsilon \mathcal{P}$, use the sharing function $f(\cdot, \cdot)$ to compute share of shares $s_{ij} = f(s_i, j)$ such that $s_i = \sum_{j \epsilon A'} r_j s_{ij}, |A'| = t' + 1$ and send s_{ij} to the corresponding $j \epsilon \mathcal{P}'$ over the private channel.
2. For each $i \epsilon \mathcal{P}$, use the verification function $E(\cdot)$ to compute $E(f(s_i, j))$ and $E(s_i)$ and reliably broadcast $\{E(s_i), E(f(s, k)) \forall k \epsilon \mathcal{P}, E(f(s_i, j)) \forall j \epsilon \mathcal{P}'\}$ to all $j \epsilon \mathcal{P}'$.
3. For each $j \epsilon \mathcal{P}'$, wait till $\geq t + 1$ reliable broadcasts containing identical values of $E(f(s, i)), i \epsilon \mathcal{P}$ complete. Store this value of $E((f(s, i))), \forall i \epsilon \mathcal{P}$ as the correct value.
4. For each $j \epsilon \mathcal{P}'$ wait till $l - t$ reliable broadcasts initiated by $i \epsilon \mathcal{I}, \mathcal{I} \subset \mathcal{P}$ complete and wait till the corresponding share of shares $s_{ij}, i \epsilon \mathcal{I}$ come in.
5. For each $j \epsilon \mathcal{P}'$, check if the following verification conditions hold for each $i \epsilon \mathcal{I}$ which has not been tested before.

$$E(s_{ij}) \equiv E(f(s_i, j))$$

$$E(s_i) \equiv \prod_{j \epsilon \mathcal{P}'} E(f(s_i, j))^{r_j}$$

$$E(s_i) \equiv E(f(s, i))$$

   Each shareholder $i \epsilon \mathcal{I}$ that passes the above verifications is added to the set X. Then reliably broadcasts the message $\{commit, X\}$.
6. For each $j \epsilon \mathcal{P}'$, wait for $l' - t'$ reliable broadcasts of $commit$ messages to complete.
7. For each $j \epsilon \mathcal{P}'$, increment $count\_phase$.
8. For each $j \epsilon \mathcal{P}'$, find the first authorized set $A, |A| = t + 1$ in $List$, such that A appears in at least $2t' + 1$ of the received $X_k, k \epsilon \mathcal{P}'$ sets. If such an authorized set A is found then propose A in a multivalued agreement protocol else propose $\{0\}$ in a multivalued agreement protocol.
9. For each $j \epsilon \mathcal{P}'$, If the multivalued agreement decides on a particular authorized set A, then
   (a) Wait for sharing and commitment messages to come in from all $i \epsilon A$.
   (b) If all $i \epsilon A$ satisfy the verification conditions then j
      i. Reconstructs share $s'_j = \sum_{i \epsilon A} r_i s_{ij}$
      ii. Computes and stores commitments

$$E(f(s, j)) = \prod_{i \epsilon A} E(f(s_i, j))^{r_i}, \forall j \epsilon \mathcal{P}'$$

      iii. Exit from the protocol.
   (c) In case the verification conditions do not pass then initiate a share recovery procedure
10. For each $j \epsilon \mathcal{P}'$, if the multivalued agreement returns 0 and $count\_phase < 2$ then wait for $t' + 1$ remaining reliable broadcasts of $commit$ messages to complete and repeat the procedure from Step 7 on wards.
11. For each $j \epsilon \mathcal{P}'$, in case an authorized set is still not decided upon then set $count\_phase = 0$. Wait for the remaining share of share and commitment messages to come in from t old shareholders, update set $\mathcal{I}$ and repeat the process from Step 5 on wards. In a maximum of two such rounds a decision on an authorized set will be reached.

Figure 5.5: Asynchronous Secret Redistribution Protocol for redistribution of s from $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$

In Step 4 the new shareholders $j \epsilon \mathcal{P}'$ wait for the scheduler to deliver the share of share and commitment messages to come from $l - t$ distinct old shareholders $i \epsilon \mathcal{I}, \mathcal{I} \subset \mathcal{P}$. In Step 5 j then tests if the following verification conditions hold for each $i \epsilon \mathcal{I}$ that has not been tested before.

$$E(s_{ij}) \equiv E(f(s_i, j)) \tag{5.1}$$

$$E(s_i) \equiv \prod_{j \epsilon \mathcal{P}'} E(f(s_i, j))^{r_j} \tag{5.2}$$

$$E(s_i) \equiv E(f(s, i)) \tag{5.3}$$

We remind the reader that equation 5.1 allows the new shareholder j to verify if s_{ij} corresponds to the sharing function $f(\cdot, \cdot)$ evaluated at s_i and j. Equation 5.2 allows the new shareholder j to verify that the commitment $E(s_i)$ corresponds to the share s_i used in the sharing function $f(s_i, j), \forall j \epsilon \mathcal{P}', \forall i \epsilon \mathcal{I}$. Equation 5.3 allows new shareholder j to verify that the share s_i corresponds to the sharing function $f(\cdot, \cdot)$ evaluated at s and i, s being the secret. Each $i \epsilon \mathcal{I}$ that satisfies the verifications is added to a set $\mathcal{X}$. j then reliably broadcasts a commit message $\{commit, X\}$.

In Step 6 each $j \epsilon \mathcal{P}'$ waits for $l' - t'$ reliable broadcasts of $commit$ messages to complete and then increments $count\_phase$ in Step 7. The basic intuition behind waiting for $l' - t'$ reliable broadcasts of $commit$ messages is to prevent the honest new shareholders from being delayed by the faulty new shareholder. We have $l' - t'$ honest new shareholders in the system. It is always possible that the faulty new shareholders purposely delay their commit messages, to increase the wait time for the honest new shareholders. To prevent an honest new shareholder from waiting for delayed messages from these faulty new shareholders, the honest new shareholders continue their protocol operations after $l' - t'$ commit messages have been received in the first phase of the protocol. Note that it is also possible that the messages from an honest new shareholder can be delayed because that particular shareholder is a slow server. In this case the other new shareholders just have to wait until the messages come in. We remind the readers of the assumption that the scheduler schedules messages such that all the messages from the honest servers arrive before the local timer set to $\mathcal{T}$

expires.

Returning to the protocol, in Step 8, j and tries to find the first authorized set A in $List$ such that A appears in at least $2t' + 1$ of the received $X_k, k\epsilon\mathcal{P}'$ sets. If j finds such an authorized set A then j proposes this authorized set in a multivalued agreement protocol as described in Figure 5.4. If j does not find such an authorized set then j proposes $\{0\}$ in the multivalued protocol.

In Step 9 j waits until a decision is reached in the multivalued agreement protocol. If the protocol decides on an authorized set A then j waits until valid share of shares arrive from each $i\epsilon A$ and reconstructs his new share s'_j and also computes and stores the commitments $E(f(s, j))$. If the multivalued agreement protocol decides on $\{0\}$ then each $j\epsilon\mathcal{P}'$ waits for the remaining t' reliable broadcasts of $commit$ messages to complete and repeats the protocol from Step 7 in Figure 5.5. In case an agreement is still not reached j sets $count\_phase = 0$. j then waits for share of share and commitment messages to come in from the remaining old shareholders and then repeats the protocol from Step 5 in Figure 5.5.

The redistribution protocol comprises of a maximum of two rounds and each round comprises of a maximum of two phases. In the best case scenario, the protocol can complete in the first round and first phase. In the worst case scenario the protocol will complete in second round and second phase.

5.3.2 Share Recovery Protocol

As in the synchronous case it is possible that after redistribution some shareholders $j\epsilon\mathcal{W}, \mathcal{W} \subset \mathcal{P}'$ hold incorrect new shares. This situation can occur when the agreed upon authorized set A used to reconstruct new shares, contains some shareholder $\overline{k}\epsilon A$ that sent incorrect share of shares to shareholders $j\epsilon\mathcal{W}$. Note that in such a situation $\overline{k}$ must have broadcast correct verification information and must have sent correct share of shares to at least $t' + 1$ honest shareholders $j\epsilon\mathcal{P}'$. To ensure that the availability of the system is intact, it is essential that the good shareholders engage in a recovery protocol which would recover their new share. To protect the availability of the system, it is essential that the shareholders engage in a recovery protocol which would recover the new share of the shareholders $j\epsilon\mathcal{W}$.

The recovery protocol is similar to the synchronous recovery protocol. This protocol differs from the synchronous recovery protocol, only in the manner of how the recovery request is sent. In the synchronous protocol we had access to a broadcast channel which was used to broadcast this recovery request. In the asynchronous setting while we have a $ReliableBroadcast$ communication primitive, this primitive is very expensive to use - $O(l'^2)$ message complexity. To avoid this high message complexity, in the recovery protocol for the asynchronous setting, the server $j \epsilon \mathcal{W}$ sends this recovery request privately to all other new shareholders using the $Send$ primitive which will result in $l' - 1$ recovery request messages for shareholder j. For the convenience of the reader we summarize the protocol again in Figure 5.6.

Share Recovery Protocol

1. For each $j \epsilon \mathcal{W}$, form a set $\mathcal{R}$ containing the identities of shareholders $\overline{k} \epsilon A$ that sent incorrect share of share $\overline{s_{kj}}$ to j. Also determine the shareholder $c \epsilon A$ that sent the correct share of share to j.
2. For each $j \epsilon \mathcal{W}$, send a recovery request $\{\mathcal{R}, c\}$ to $i \epsilon \mathcal{P}', i \neq j$
3. For each $i \epsilon (\mathcal{P}' - \mathcal{W})$, on receiving a recovery request from shareholder j, compute $\delta_{kc}^i = s_{ki} + s_{ci}, \forall k \epsilon \mathcal{R}$, and send $\{\delta_{kc}^i\}$ to shareholder j over the private channel.
4. For each $j \epsilon \mathcal{W}$, on receiving the recover share of shares from shareholder i, verify that
$$E(\delta_{kc}^i) \equiv E(f(s_k, i))E(f(s_c, i))$$
5. For each $j \epsilon \mathcal{W}$, wait until receive recovery share of shares from $t' + 1$ shareholders that satisfy the above verification condition. (Let $\mathcal{C}$ denote this set of shareholders).
6. For each $j \epsilon \mathcal{W}$, recover share of share s_{kj} as follows
   (a) First compute $s_k + s_c = \sum_{i \epsilon \mathcal{C}} r_i \delta_{kc}^i$
   (b) Then compute $\delta_{kc}^j = f(s_k + s_c, j)$
   (c) Finally compute $s_{kj} = \delta_{kc}^j - s_{cj}$
7. For each $j \epsilon \mathcal{W}$, after recovering share of shares $s_{kj}, k \epsilon \mathcal{R}$, recover new share $s'_j = \sum_{k \epsilon A} r_k s_{kj}$.

Figure 5.6: Share Recovery Protocol

5.4 Analysis

In this section we analyze our secret redistribution protocol. We show that the redistribution protocol satisfies the properties of *Liveness*, *Robustness*. The *Correctness*, *Secrecy, Flexibility* properties are the same as proved in the synchronous analysis section. Also note that the analysis for the share recovery protocol as described before for synchronous systems also holds for asynchronous systems.

Liveness

Theorem 10 *If the scheduler activates all honest shareholders and delivers all associated messages within a redistribution period, then all honest shareholders complete the redistribution except with negligible probability.*

Proof: We have to prove that if the scheduler delivers all associated messages within a redistribution period, then each honest shareholder will eventually complete all protocol operations and complete the redistribution protocol. If the scheduler delivers all the messages from honest old shareholders, each honest new shareholder will reliably broadcast a commit message on the same set X to all other new shareholders. If the scheduler delivers all the reliable broadcasts of the commit messages sent by honest new shareholders, then each honest new shareholder will propose the same authorized set A in a multivalued agreement. It follows that since at least $l' - t'$ honest new shareholders have entered the multivalued agreement with the same authorized set A, all honest new shareholders will decide on the proposed set A and consequently complete the redistribution.

Robustness

Robustness implies correctness in the presence of faulty shareholders. We first prove some basic Lemmas and then move on to prove that our redistribution protocol for asynchronous systems is robust.

Lemma 9 *Suppose a faulty old shareholder $\overline{i} \epsilon \mathcal{P}$ sends incorrect share of shares $\overline{s_{ij}}$, to any correct new shareholder $j \epsilon \mathcal{P}'$ then $\overline{i}$ would not be present in set X_j*

Proof: When a correct shareholder j receives the share of shares and commitments from shareholder $\overline{i}$, j verifies if the information sent by $\overline{i}$ passes Equation 5.1, Equation 5.2 and Equation 5.3. Only if $\overline{i}$ passes all checks, $\overline{i}$ would be added to set X_j. Since $\overline{i}$ has sent incorrect share of share to j Equation 5.1 would not pass for $\overline{i}$ and consequently $\overline{i}$ would not be added to set X_j.

Lemma 10 *Suppose a faulty old shareholder $\overline{i} \epsilon \mathcal{P}$ reliably broadcasts incorrect commitment $\overline{E(f(s,k))}$, $\forall k \epsilon \mathcal{P}$ then $\overline{i}$ would not be present in set X_j, for all correct $j \epsilon \mathcal{P}'$.*

Proof: We remind the readers of our assumption that there are at least $t+1$ old shareholders that are correct and a maximum of t are faulty. Thus in this case after receiving the same commitment value

$E(f(s,k))$ from $t+1$ or more old shareholders the new shareholder can arrive at the correct value of $E(f(s,k)), \forall k \epsilon \mathcal{P}$. Each new shareholder can thus identify the faulty old shareholder $\overline{i}$ based on the inconsistent values of $\overline{E(f(s,k))}$. Consequently each correct new shareholder j would not add $\overline{i}$ to set X_j.

Lemma 11 *Suppose $\overline{i}$ reliably broadcasts an incorrect commitment to share $\overline{E(s_i)}$, then $\overline{i}$ would not be present in X_j for all correct $j \epsilon \mathcal{P}'$.*

Proof: In this case $\overline{E(s_i)}$ would not pass Equation 5.3 and consequently all correct $j \epsilon \mathcal{P}'$ would not add $\overline{i}$ to X_j.

Lemma 12 *Suppose $\overline{i}$, sends share of share $\overline{s_{ij}}$ and broadcasts a commitment $\overline{E(f(s_i,j))}$ which satisfy Equation 5.1, but $\overline{i}$ has used an incorrect share $\overline{s_i}$ when creating $\overline{s_{ij}}$, then $\overline{i}$ would not be present in set X_j,for all correct $j \epsilon \mathcal{P}'$.*

Proof: In this case $\overline{E(f(s_i,j)}$ when tested locally by each shareholder $j \epsilon \mathcal{P}'$ will not pass the verification Equation 5.2 and Equation 5.1 simultaneously. Consequently a correct new shareholder $j \epsilon \mathcal{P}'$ would not add $\overline{i}$ to X_j.

Lemma 13 *Suppose a faulty new shareholder $\overline{j} \epsilon \mathcal{P}'$ commits on a set X containing a faulty old shareholder $\overline{i} \epsilon \mathcal{P}$ who has sent incorrect share of shares to some new shareholders, then $A, \overline{i} \epsilon A, A \subset X$ may or may not be chosen for reconstruction.*

Proof: In this case if shareholder k gets $\geq 2t'+1$ commits on A then k will propose A for a multivalued agreement. In case shareholder k does not receive as many commits on A, then k will not propose A for a multivalued agreement. If the multivalued agreement decides on A then A would be chosen for reconstruction, otherwise k would chose another authorized set A from the $List$ to agree upon for reconstruction.

Theorem 11 *As long as there are $t+1$ correct old shareholders and $2t'+1$ correct new shareholders the correctness of the redistribution will be preserved in the presence of faulty shareholders.*

Proof: From Lemmas 9, 10, 11 and 12 it follows that a correct shareholder $j \epsilon \mathcal{P}'$ would not broadcast a commit on $X, \overline{i} \epsilon X$ if $\overline{i}$ has sent incorrect share of share or incorrect commitment information to j. From the protocol description one can deduce that an authorized set $A, \overline{i} \epsilon A$ will be proposed

for agreement only if $\bar{i}$ has sent correct information to at least $t' + 1$ correct new shareholders. From Lemma 13 we can see that even in the presence of incorrect commit messages broadcast by faulty new shareholders the correct new shareholders will eventually reach an agreement on a correct authorized set A. Since there are at least $t + 1$ correct old shareholders, we can be ensured that there would definitely be one correct authorized set which can be used for reconstruction of the new share. Further since there are at least $2t' + 1$ correct new shareholders we can be ensured that $2t' + 1$ commit will definitely be received on this authorized set $A, A \subset X$, and after redistribution at least $t' + 1$ correct shareholders would hold valid shares. Since there are at least $t' + 1$ correct shareholders would have valid shares after redistribution, the robustness property of the recovery protocol as proved in Theorem 7 would hold, and after recovery all correct shareholders would have correct shares. Consequently it follows that the correctness of the redistribution would be preserved even in the presence of faulty shareholders.

Theorem 12 *The new shareholders definitely reach an agreement on an authorized set A, in phase two of round two of the redistribution protocol*

Proof: In round two and phase two of the redistribution protocol, each new shareholder would have received the sharing and the commitment messages from all honest old shareholders. In addition each new shareholder would have received the reliable broadcasts of the set X from all honest new shareholders. It then follows from the Liveness proof in Theorem 10 that the new shareholders will reach and agreement on an authorized set A.

Chapter 6

Implementation of Dynamic Secret Redistribution in Asynchronous Systems

6.1 Asynchronous Dynamic Secret Redistribution with Pedersen VSS

In this section we describe a Dynamic Secret Redistribution protocol, with Pedersen's Verification, for Asynchronous systems.

6.1.1 Notation

We adopt the same notation we proposed in the synchronous implementation. We enumerate the basic points again for the convenience of the reader.

1. p and q are large primes such that $q|p-1$. G_q is a unique subgroup of $\mathbb{Z}_p^*$ of order q, g is a generator of G_q, and h is an element of G_q such that nobody knows $log_g h$.

2. The sharing function f is Shamir's polynomial sharing method and the reconstruction function r is obtained through Lagrange's interpolation. In addition the shareholders verify their shares using Pedersen's verification scheme.

3. In Pedersen's scheme the dealer commits himself to the secret $s \epsilon \mathbb{Z}_q$ by choosing $u \epsilon \mathbb{Z}_q$ at random and computing $E(s,u) = g^s h^u$

6.1.2 Dynamic Update Protocol

The dynamic update protocol comprises of a secret redistribution protocol and may also include a share recovery protocol. As mentioned in our model, the redistribution from $(t+1, l)$ threshold scheme to $(t'+1, l')$ threshold scheme is initiated when the number of servers compromised by the adversary reaches the threshold α or when the requirements of the system have changed. After redistribution honest shareholders that have received incorrect share of shares from old shareholders $\overline{i} \epsilon A$, A being the authorized set, initiate a share recovery protocol which recovers their share of shares and consequently recovers their new share.

Secret Redistribution Protocol

This protocol redistributes from access structure $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to access structure $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$. The protocol is detailed in Figure 6.1. We assume that initially each shareholder $i \epsilon \mathcal{P}$ holds a valid sharing and also stores the commitments $E_m = E(f_m, v_m), m \epsilon \{0 \dots t\}$ where f_m and v_m are coefficients of the polynomial chosen initially to distribute shares of the secret s.

As described in the model in Chapter 5 each shareholder $j \epsilon \mathcal{P}'$ initializes a local variable $count\_phase = 0$ which denotes the current phase of the protocol. We also assume that all $j \epsilon \mathcal{P}'$ have an enumerated list $List$ of authorized sets chosen from $\mathcal{P}$. This $List$ is the same for all correct $j \epsilon \mathcal{P}'$. Each shareholder has two local sets X and $\mathcal{I}$. X is used to store the identities of the old shareholders that pass the verification equations. $\mathcal{I}$ is used to store the identities of those old shareholders whose messages have been received. In addition we assume each new shareholder is equipped with a local timer which is set to a specific time out $\mathcal{T}$. We assume that the scheduler schedules messages such that all the messages from the honest servers arrive before the timer expires. In case such scheduling is not possible, then the new shareholders just have to wait till the messages come in from the honest old shareholders. Redistribution proceeds along the lines of the general protocol described in Chapter 5 and is summarized in Figure 6.1.

Share Recovery Protocol

The recovery protocol is on the same lines as described in the asynchronous model in Chapter 5. For the convenience of the reader we summarize the protocol in Figure 6.2. Note that $\mathcal{W} \subset \mathcal{P}'$

Dynamic Asynchronous Secret Redistribution protocol for Shamir's Secret Sharing and Pedersen's Verification

1. For each $i \epsilon \mathcal{P}$, use the polynomial $f'_i(j) = s_i + f'_{i1} j + \ldots + f'_{it'} j^{t'}$ to compute share of shares $s_{ij} = f'_i(j)$ and use polynomial $v'_i(j) = u_i + v'_{i1} j + \ldots + v'_{it'} j^{t'}$ to compute $u_{ij} = v'_i(j)$. Send (s_{ij}, u_{ij}) to corresponding $j \epsilon \mathcal{P}'$ over the private channel.
2. For each $i \epsilon \mathcal{P}$, compute $E'_{ik} = E(f'_{ik}, v'_{ik}) = g^{f'_{ik}} h^{v'_{ik}}$, $k = 0, \ldots t'$ and reliably broadcast $\{E_m, E'_{ik}\}$ to all $j \epsilon \mathcal{P}'$.
3. For each $j \epsilon \mathcal{P}'$, wait till $\geq t + 1$ reliable broadcasts containing identical values of $E_m, m \epsilon \{0, \ldots, t\}$ to complete. Store these values of E_m as the correct values.
4. For each $j \epsilon \mathcal{P}'$ wait till $l - t$ reliable broadcasts initiated by $i \epsilon \mathcal{I}, \mathcal{I} \subset \mathcal{P}$ complete and wait till the corresponding share of shares $(s_{ij}, u_{ij}), i \epsilon \mathcal{I}$ come in.
5. For each $j \epsilon \mathcal{P}'$, check if the following verification conditions hold for each $i \epsilon \mathcal{I}$ which has not been tested before.

$$E(s_{ij}, u_{ij}) = \prod_{k=0}^{t'} (E'_{ik})^{j^k} \tag{6.1}$$

$$E(s_i, u_i) = E'_{i0} = \prod_{m=0}^{t} (E_m)^{i^m} \tag{6.2}$$

   Each shareholder $i \epsilon \mathcal{I}$ that passes the above verifications is added to the set X. Then reliably broadcasts the message $\{commit, X\}$.
6. For each $j \epsilon \mathcal{P}'$, wait for $l' - t'$ reliable broadcasts of $commit$ messages to complete.
7. For each $j \epsilon \mathcal{P}'$, increment $count\_phase$.
8. For each $j \epsilon \mathcal{P}'$, find the first authorized set $A, |A| = t + 1$ in $List$, such that A appears in at least $2t' + 1$ of the received $X_k, k \epsilon \mathcal{P}'$ sets. If such an authorized set A is found then propose A in a multivalued agreement protocol else propose $\{0\}$ in a multivalued agreement protocol.
9. For each $j \epsilon \mathcal{P}'$, If the multivalued agreement decides on a particular authorized set A, then
   (a) Wait for sharing and commitment messages to come in from all $i \epsilon A$.
   (b) If all $i \epsilon A$ satisfy the verification conditions then j
      i. Reconstructs share $s'_j = \sum_{i \epsilon A} r_i s_{ij}, u'_j = \sum_{i \epsilon A} r_i u_{ij}$
      ii. Compute and store commitments

$$E_k = \prod_{i \epsilon A} (E'_{ik})^{r_i}, k = 0 \ldots t' \tag{6.3}$$

         where r_i is determined using Lagrange's Interpolation.
      iii. Exit from the protocol.
   (c) In case the verification conditions do not pass then initiate a share recovery procedure
10. For each $j \epsilon \mathcal{P}'$, if the multivalued agreement returns 0 and $count\_phase < 2$ then wait for $t' + 1$ remaining reliable broadcasts of $commit$ messages to complete and repeat the procedure from Step 7 on wards.
11. For each $j \epsilon \mathcal{P}'$, in case an authorized set is still not decided upon then set $count\_phase = 0$. Wait for the remaining share of share and commitment messages to come in from t old shareholders, update set $\mathcal{I}$ and repeat the process from Step 5 on wards. In a maximum of two such rounds a decision on an authorized set will be reached.

Figure 6.1: Asynchronous Secret Redistribution Protocol for redistribution of s from $\Gamma_{\mathcal{P}}^{(t+1,l)}$ to $\Gamma'^{(t'+1,l')}_{\mathcal{P}'}$

denotes the set of shareholders that need recovery.

Share Recovery Protocol

1. For each $j \epsilon \mathcal{W}$, form a set $\mathcal{R}$ containing the identities of shareholders $\overline{k} \epsilon A$ that sent incorrect share of share $\overline{s_{kj}}$ to j. Also determine the shareholder $c \epsilon A$ that sent the correct share of share to j.
2. For each $j \epsilon \mathcal{W}$, send a recovery request $\{\mathcal{R}, c\}$ to $i \epsilon \mathcal{P}', i \neq j$.
3. For each $i \epsilon (\mathcal{P}' - \mathcal{W})$, on receiving a recovery request from shareholder j, compute $\delta^i_{kc} = s_{ki} + s_{ci}; \gamma^i_{kc} = u_{ki} + u_{ci}, \forall k \epsilon \mathcal{R}$, and send $\{\delta^i_{kc}, \gamma^i_{kc}\}$ to shareholder j over the private channel.
4. For each $j \epsilon \mathcal{W}$, on receiving the recover share of shares from shareholder i, verify that

$$E(\delta^i_{kc}, \gamma^i_{kc}) \equiv \prod_{m=0}^{t'} (E'_{km})^{i^m} \prod_{m=0}^{t'} (E'_{cm})^{i^m} \tag{6.4}$$

5. For each $j \epsilon \mathcal{W}$, wait until receive recovery share of shares from $t' + 1$ shareholders that satisfy the above verification condition. (Let $\mathcal{C}$ denote this set of shareholders).
6. For each $j \epsilon \mathcal{W}$, recover share of share s_{kj} as follows
   (a) First $\delta^j_{kc} = s_{kj} + s_{cj}$ by interpolating the recovery share of shares

$$\delta^j_{kc} = \sum_{i \epsilon \mathcal{C}} r_i \delta^i_{kc}, r_i = \prod_{m \epsilon \mathcal{C}, m \neq i} \frac{m - j}{m - i} \tag{6.5}$$

   (b) Then compute $s_{kj} = \delta^j_{kc} - s_{cj}$

   Similarly recover share of share u_{kj}.
7. For each $j \epsilon \mathcal{W}$, after recovering share of shares $s_{kj}, k \epsilon \mathcal{R}$, recover new share $s'_j = \sum_{k \epsilon A} r_k s_{kj}, u'_j = \sum_{k \epsilon A} r_k u_{kj}$ where r_k is determined using Lagrange's Interpolation function. .

Figure 6.2: Share Recovery Protocol

Complexity

In this section we evaluate the redistribution protocol on the basis of message complexity and computational complexity.

1. Message Complexity: During redistribution each old shareholder sends the shares and commitment messages to each new shareholder. This accounts for $O(ll'^2)$ messages. In addition each new shareholder reliably broadcasts the set X. This accounts for $O(l'^3)$ messages. Finally each new shareholder participates in a multivalued agreement whose message complexity is $O(l'^3)$. Thus the expected message complexity of the redistribution protocol is $O(l'^3)$. During recovery each new shareholder that needs recovery sends a recovery request to all other shareholders. This corresponds to $l' - 1$ recovery request messages. In response to the recovery request each honest new shareholder sends recovery share of shares to the

shareholder requesting for recovery. This accounts for a maximum of $l' - 1$ messages. Thus for each server that needs recovery we have a maximum of $2l' - 2$ recovery messages.

2. Computational Complexity: The main factor to be considered here is the cost of computing the verifications Equations 6.1 6.2 and 6.4 and the cost of computing the commitments in Equation 6.3. Excluding the cost of computing each commitment $E(a, b) = g^a h^b$, the verification conditions results in $O(tt')$ multiplications and $O(tt')$ exponentiations. In the worst case Equation 6.1 and Equation 6.2 would have to be computed l times. In the worst case Equation 6.4 would have to be computed $l - 1$ times. Equation 6.3 will have to computed once.

6.2 Results

We implemented the redistribution protocol using Shamir's secret sharing and Pedersen's verification as described in the previous section. We used the Open SSL Big Number Library for performing the modular arithmetic operations involved in the protocol. In order to incorporate asynchrony into the system, we introduced random delays which were chosen from data collected by Sariou *et al.* [SGG02]. We allowed the user to vary the various redistribution parameters and evaluated our protocol based on Computational Cost of verifications and End to End Cost of the protocol. The results of our tests are depicted in the following graphs

In Figure 6.3 we plot Verification Cost against Share Size. We vary the share size from 16 bits to 1024 bits and plot the cost of computing the Equations 6.1, 6.2 and 6.3 while redistributing from access structure $\Gamma_{\mathcal{P}}^{(2,4)}, |P| = 4$ to access structure $\Gamma'^{(2,4)}_{\mathcal{P}'}, |P'| = 4$. It is observed that the verification cost increases linearly with share size. This is expected because with increase in share size the cost of computing Equation 6.1 ,Equation 6.2 and 6.3 will increase linearly, since the modular arithmetic operations have to be performed over larger numbers.

In Figure 6.4 we plot Verification Cost against Threshold. We vary the threshold from $t' + 1 = 2$ to $t' + 1 = 5$ and plot the cost of computing Equations 6.1, 6.2, and 6.3. The group size $l' = 13$ and the share size is kept constant at 512 bits. It is observed that as threshold increases the cost of

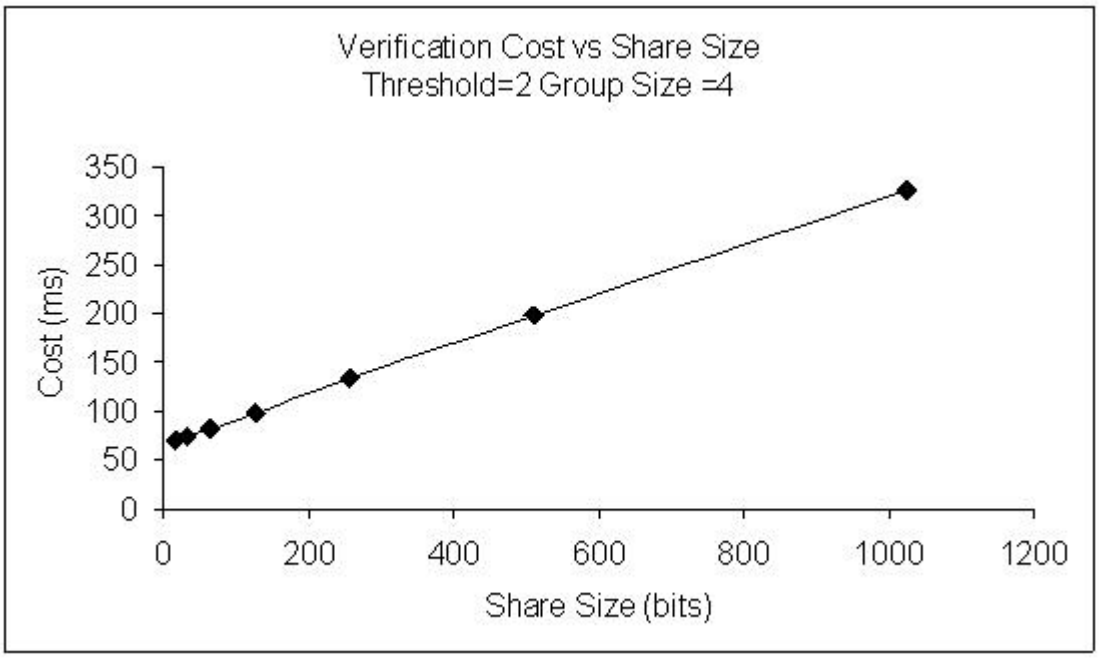

Figure 6.3: Verification Cost vs Share Size

verification also increases linearly. This trend is expected because with increase in threshold, the number of exponentiations and multiplications that need to be performed in Equations 6.1,6.2 and 6.3 increases linearly.

In Figure 6.5 we plot Verification cost of computing Equations 6.1, 6.2, and 6.3 against group size. The threshold is kept constant at $t + 1 = 2$, share size is kept constant at 512 bits and group size is varied from $l = 4$ to $l = 7$. It is seen that as group size increases the cost of verification also increases. This is expected. Consider the first round of the protocol in Figure 6.1. It is seen that before entering the multivalued agreement each shareholder verifies the share of shares sent by $l - t$ old shareholders using Equations 6.1 6.2. Since t is kept constant and l is increased, it can be deduced that the number of times each shareholder needs to perform these verifications increases with l and consequently the verification cost increases.

In Figure 6.6 we plot Recovery Cost against Threshold. This is the cost of computing the Equation 6.4. In the experiment we kept one faulty old share holder $\bar{i}$ who sent an incorrect share of share $\overline{s_{ij}}$ to one correct new share holder j. The cost in the graph corresponds to the cost incurred by

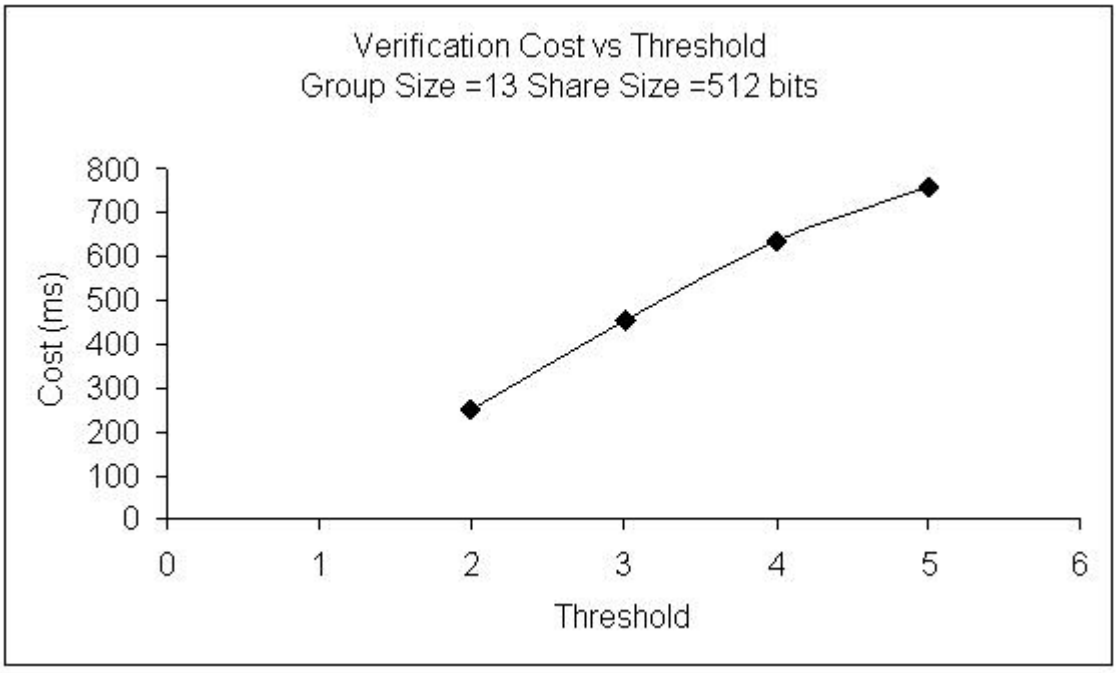

Figure 6.4: Verification Cost vs Threshold

j when verifying the recovery share of shares corresponding to share of share $\overline{s_{ij}}$. We varied the threshold from $t' + 1 = 2$ to $t' + 1 = 5$, plot the cost of computing the Equation 6.4. The group size is kept constant at $l' = 13$ and the share size is kept constant at 512 bits. It is observed that as the threshold increases the recovery cost increases linearly. This trend is expected because with increase in threshold, the number of exponentiations and multiplications that need to be performed in Equation 6.4 increases linearly.

In Figure 6.7 we plot End to End Cost against Share size. End to End Cost is the time taken for the redistribution protocol to complete. This is measured starting from the point when the old shareholders compute share of shares to the point when the new shareholders reconstruct their new share. This includes Verification Cost and also includes Network Delay cost. Since the system is asynchronous and the message complexity is high the network delay cost is very high. We kept threshold and group size constant at 2 and 4 respectively and varied share size from 16 bits to 1024 bits. It is seen that the end to end cost increases linearly with share size. End to End cost includes the verification cost of the Equations 6.1,6.2 and 6.3. Since verification cost increases with share size as depicted in Figure 6.3, it follows that end to end cost will increase with share size.

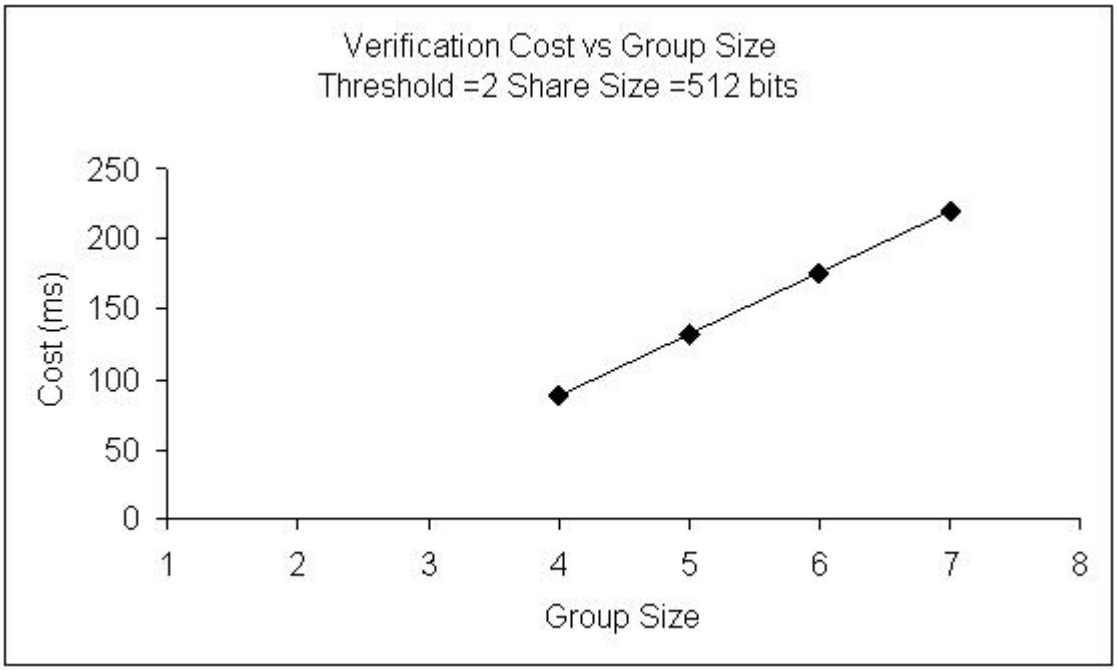

Figure 6.5: Verification Cost vs Group size

In Figure 6.8 we plot End to End Cost against Threshold. The group size is kept constant at 7 and share size is kept constant at 512 bits. We vary the threshold from $t+1=1$ to $t+1=3$. It is seen as threshold increases end to end cost also increases. This is expected. Since End to End Cost includes the Verification Cost, and since Verification Cost increases with threshold as depicted in Figure 6.4, it follows that End to End Cost will increase with threshold.

In Figure 6.9 we plot End to End Cost against Group Size. The threshold is kept constant at 2 and the share size is kept constant at 512 bits. We vary the group size from $l=4$ to $l=7$. It is observed that end to end cost increases linearly with group size. This is expected because as the group size increases the number of messages in the system increase and consequently there are more number of delays in the system and consequently the end to end cost increases.

In Figure 6.10 we plot End to End Cost vs Number of Recovery Requests. The threshold and group size are kept constant at $t'+1=2$ and $l'=7$ respectively. Share Size is kept constant at 256 bits. The End to End Cost plotted includes the Verification Cost, the Recovery Cost and the network delay cost. It is observed that end to end cost increases linearly with the number of recovery requests. This

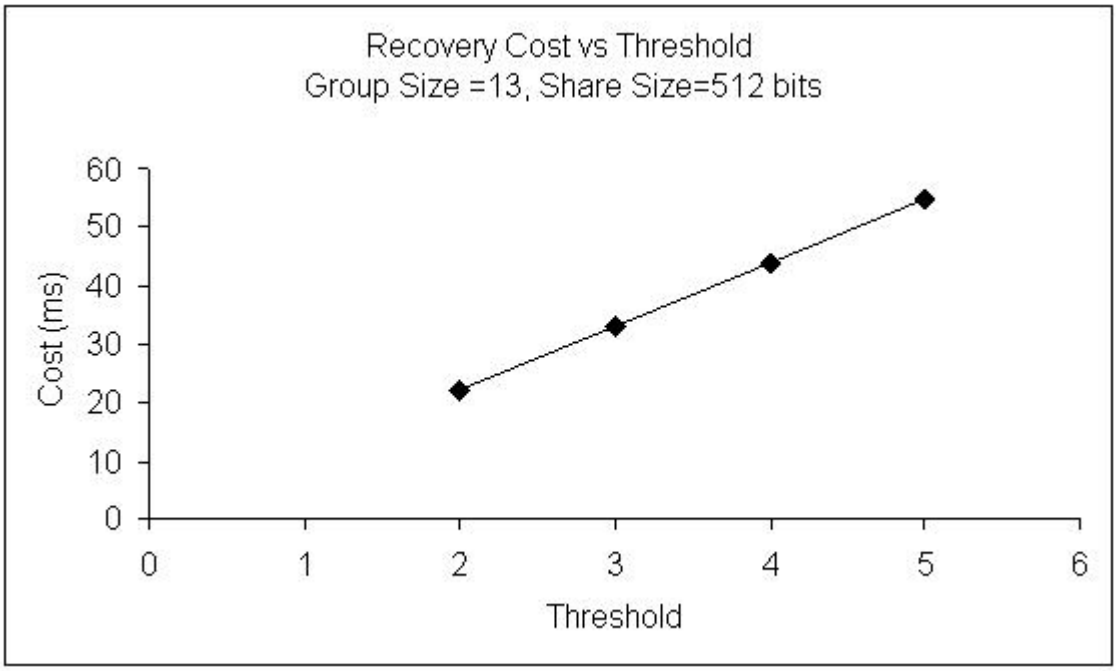

Figure 6.6: Recovery Cost vs Threshold

is expected, since, as the number of recovery requests increase, the number of recovery messages in the system increases, which increases the network delay cost and consequently the end to end cost increases.

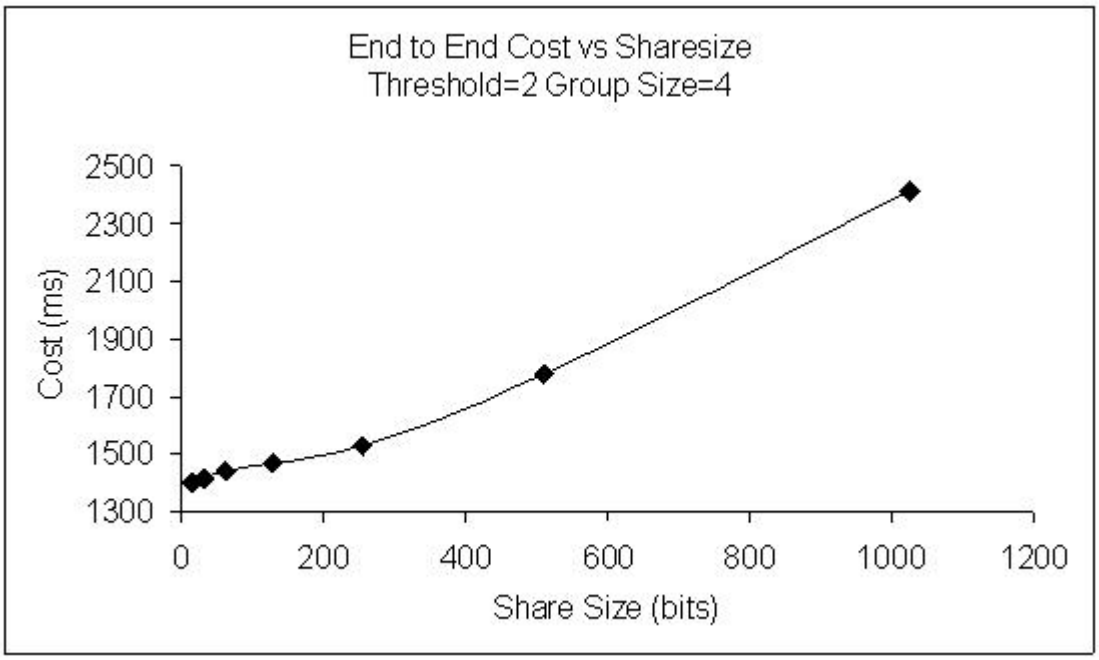

Figure 6.7: End to End Cost vs Share size

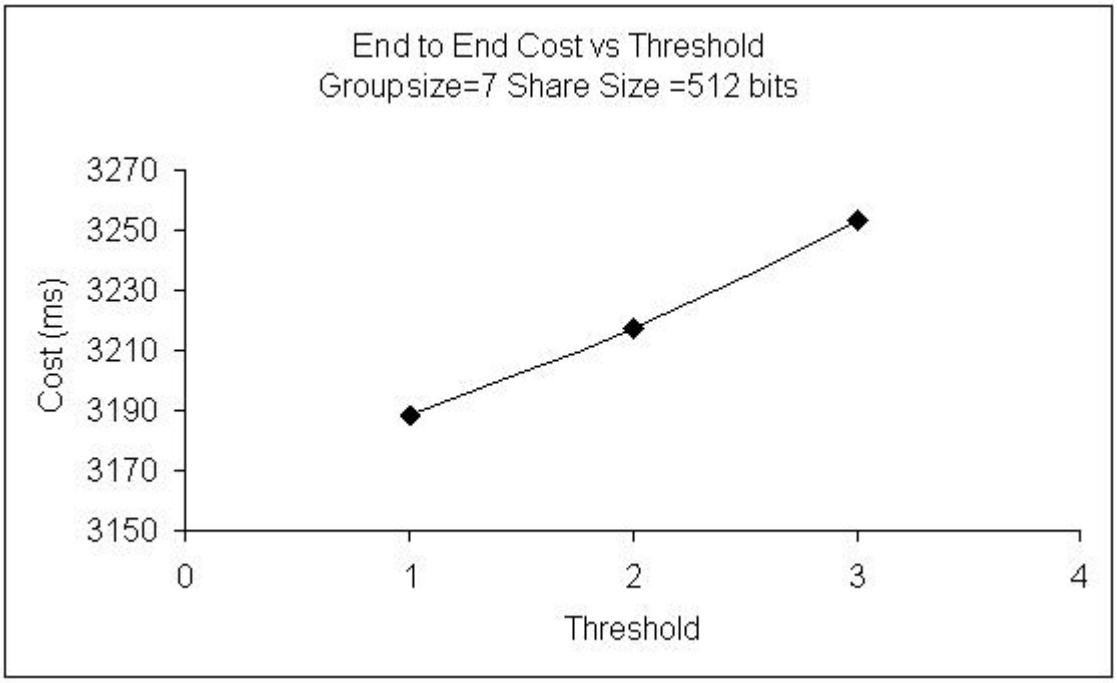

Figure 6.8: End to End Cost vs Threshold

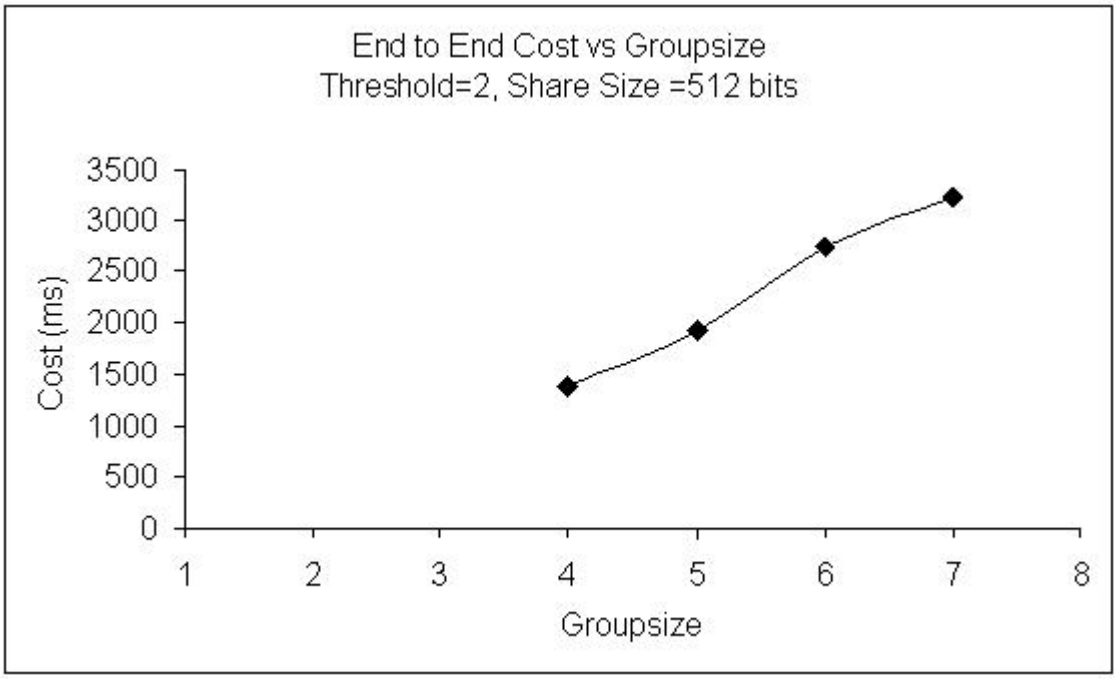

Figure 6.9: End to End Cost vs Group size

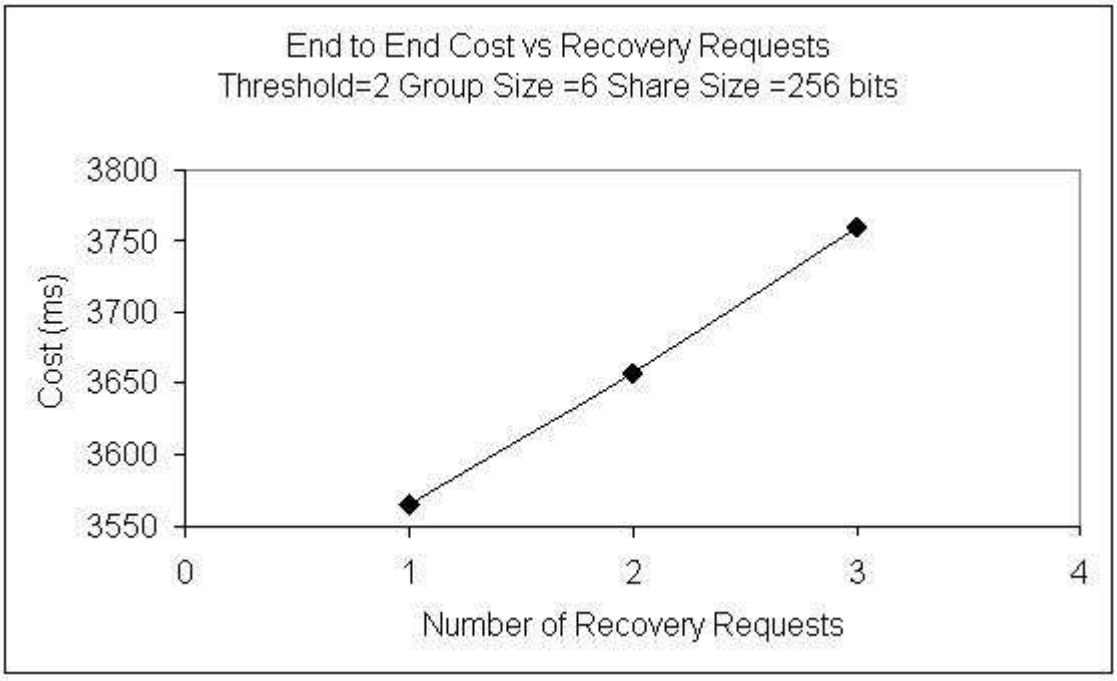

Figure 6.10: End to End Cost vs Number of Recovery Requests

Chapter 7

Summary

In this thesis we identified the drawbacks of traditional proactive schemes. We introduced the notion of a *flexible mobile adversary* – an adversary whose compromise rate can change with time. We point out that traditional proactive schemes cannot tolerate such an adversary. Traditional proactive schemes do not allow for change in group size and threshold. These limitations motivate the need for a new way of secret sharing which we call *Dynamic Secret Redistribution*.

We proposed a definition, model and a general protocol for *Dynamic Secret Redistribution* in synchronous and asynchronous systems. We present a detailed analysis of our general protocols for synchronous and asynchronous systems and prove that they meet the properties enumerated in the definition. We implement these protocols using Shamir's secret sharing and Pedersen's Verification and analyze their efficiency in terms of Message Complexity , Computational Complexity and End to End Cost of the protocol.

Bibliography

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, volume 48, pages 313–317, June 1979.

[Bra84] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, 1984.

[BT83] Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 12–26. ACM Press, 1983.

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *Proceedings of the 26th IEEE Annual Symposium on Foundations of Computer Science*, pages 383–395, 1985.

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 88–97. ACM Press, 2002.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. *Lecture Notes in Computer Science*, 2139:524–541, 2001.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 123–132. ACM Press, 2000.

[Coh87] Josh Cohen Benaloh. Secret sharing homomorphisms: keeping shares of a secret secret. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 251–260. Springer-Verlag, 1987.

[Des94] Yvo Desmedt. Threshold cryptography. *European Transactions on Telecommunication*, 5:449–457, 1994.

[Des97] Yvo Desmedt. Some recent research aspects of threshold cryptography. In *Information Security, First International Workshop ISW '97*, volume 1396 of *Lecture Notes in Computer Science*, pages 158–173. Springer-Verlag, 1997.

[DJ97] Y. Desmedt and S. Jajodia. Redistributing secret shares to new access structures and its applications. Technical Report ISSE TR-97-01, George Mason University,Fairfax,VA, July 1997.

[Fel87] Paul Feldman. A practical scheme for non- interactive verifiable secret sharing. In *Proceedings of the 28th FOCS, IEEE 1987*, pages 427–437, 1987.

[FGMY97] Y. Frankel, P. Gemmel, P. MacKenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *Proceedings of 38th IEEE Ann.Symp.On Foundations of Computer Science*, pages 384–393, October 1997.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to prove all np statements in zero knowledge and a methodology of cryptographic protocol design. In *Proceedings of CRYPTO 1986, the 6th Annual International Cryptology Conference*, volume 263 of Lecture Notes in Computer Science, pages 171–185, 1987.

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing. In *Proceedings of CRYPTO,the 15th Ann.Intl Cryptology Conf.*, volume 963 of Lecture Notes in Computer Science, pages 339–352, August 1995.

[Jar95] Stanislaw Jarecki. Proactive secret sharing and public key cryptosystem. Master's thesis, 1995.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59. ACM Press, 1991.

[Ped92] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advanced in Cryptology, CRYPTO 1991.*, volume 576 of Lecture Notes in Computer Science, pages 129–140, 1992.

[Rab98] T. Rabin. A simplified approach to threshold and proactive rsa. In *Proceedings of CRYPTO 1998, the 18th Annual International Cryptology Conference*, volume 1462 of Lecture Notes in Computer Science, pages 89–104, 1998.

[SGG02] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[Sha79] A. Shamir. How to share a secret. *Communications of ACM*, (22(11)):612–613, November 1979.

[Tou84] Sam Toueg. Randomized byzantine agreements. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 163–178. ACM Press, 1984.

[WWW02] T. Wong, C. Wang, and J. Wing. Verifiable secret redistribution for threshold sharing schemes. In *Proceedings of 1st IEEE Security in storage workshop Greenbelt M.D.*, December 2002.

[Zho01] Lidong Zhou. *Towards Building Secure and Fault-tolerant On-line Services*. PhD thesis, Computer Science Department, Cornell University, Ithaca, NY USA, May 2001.