Decentralized Recovery for Survivable Storage Systems

Theodore Ming-Tao Wong

May 2004

CMU-CS-04-119

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:

Jeannette M. Wing, Chair (Dept. of Computer Science, Carnegie Mellon University)
Gregory R. Ganger (Dept. of Electrical and Computer Engineering, Carnegie Mellon University)
Chenxi Wang (Dept. of Electrical and Computer Engineering, Carnegie Mellon University)
Michael K. Reiter (Dept. of Electrical and Computer Engineering, Carnegie Mellon University)

This research is sponsored in part by the Army Research Office (contract DAAD19-01-1-0485), the Defense Advanced Research Projects Agency (AF contracts F30602-99-2-0539-AFRL, F33615-93-1-1330, F30602-97-2-0031), and the National Science Foundation (contracts CCR-0121547, CCR-0208853, CCR-9523972).

Keywords: Survivable storage systems, verifiable secret redistribution, threshold sharing

To my mother and father, who taught me the importance of scholarship.

Abstract

Modern society has produced a wealth of data to preserve for the long term. Some data we keep for cultural benefit, in order to make it available to future generations, while other data we keep because of legal imperatives. One way to preserve such data is to store it using *survivable* storage systems. Survivable storage is distinct from reliable storage in that it tolerates confidentiality failures in which unauthorized users compromise component storage servers, as well as crash failures of servers. Thus, a survivable storage system can guarantee both the availability and the confidentiality of stored data.

Research into survivable storage systems investigates the use of m-of-n threshold sharing schemes to distribute data to servers, in which each server receives a share of the data. Any m shares can be used to reconstruct the data, but any $m-1$ shares reveal no information about the data. The central thesis of this dissertation is that to truly preserve data for the long term, a system that uses threshold schemes must incorporate recovery protocols able to overcome server failures, adapt to changing availability or confidentiality requirements, and operate in a decentralized manner.

To support the thesis, I present the design and experimental performance analysis of a *verifiable secret redistribution* protocol for threshold sharing schemes. The protocol redistributes shares of data from old to new, possibly disjoint, sets of servers, such that new shares generated by redistribution cannot be combined with old shares to reconstruct the original data. The protocol is decentralized, and does not require intermediate reconstruction of the data; thus, one does not create a central point of failure or risk the exposure of the data during protocol execution. The protocol incorporates a verification capability that enables new servers to confirm that their shares can be used to reconstruct the original data.

Acknowledgements

I think no student could find a better advisor than in mine, Jeannette Wing. She has always been there to provide feedback, support, and guidance throughout the course of my dissertation research, and I thank her for all of her efforts. I also thank the members of my committee—Greg Ganger, Mike Reiter, and Chenxi Wang—for their feedback and support.

I would also like to acknowledge the guidance that I have received from my colleagues in the research community. In particular, I would like to thank Garth Gibson (my previous advisor), Richard Golding, and John Wilkes for their research and personal advice.

I am grateful to both Ken Birman (my M.Eng. advisor) and Fred Schneider at Cornell University for providing lab facilities and research feedback while I was in Ithaca for a month in 2002. I also thank the members of the PDL Consortium (including EMC, Hewlett-Packard, Hitachi, IBM, Intel, LSI, Microsoft, Network Appliance, Oracle, Panasas, Seagate, Sun, and Veritas) for their feedback and support.

I would like to thank Paul Mazaitis and Joan Digney for taking the time to proofread my dissertation for any bugs in the writing, as well as for all of their help during PDL Retreats and Open Houses.

I have been fortunate to have been a part of the CMU SCS community. Not only is it a top-notch research environment, but it has also been a friendly and welcoming place to me. I have made many friends while at CMU, and I thank them all for their support, and for being a part of my life and letting me be a part of theirs.

Last, but not least, I would to thank my wife, Addie, for her love and encouragement. She never stopped believing that I could (and would) finish my dissertation, even when I had my doubts.

And, for anyone who has been on `zephyr` in the last two years, “<slip>”.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern society has produced a wealth of data to preserve for the long term. Some data we keep for cultural benefit, in order to make it available to future generations. For example, the Internet Archive (`http://www.archive.org`) aims to preserve indefinitely both the contents of all Internet websites and of all digitized physical media. Other data we keep because of legal imperatives. For example, several laws (*e.g.*, the Gramm-Leach-Bliley Act of 1999 and the Sarbanes-Oxley Act of 2002) mandate retention and privacy standards for financial records.

One way to preserve long-term data is to store it using *survivable* storage systems. Survivable storage systems are generally aggregations of unreliable components, such as heterogeneous "bricks" able to provide both storage and metadata management functions [22, 34], or peer-to-peer workstations [1, 46]. They are distinct from reliable storage systems [40] in that they tolerate confidentiality failures (in which unauthorized users compromise components) as well as crash failures of components. Thus, survivable storage systems are able to guarantee both availability (will one be able to recover one's data?) and confidentiality (can one be sure that an unauthorized person has not obtained one's data?).

A number of researchers propose using *m-of-n threshold sharing schemes* [49] in survivable storage systems to tolerate component failures. A system that uses a threshold scheme distributes n *shares* of the original data to components such that any m can be used to reconstruct the data. Moreover, any $m-1$ shares reveal no information about the data. Thus, the system can tolerate the loss $n-m$ shares without losing data availability, and the compromise of $m-1$ shares (*i.e.*, revelation to an unauthorized user) without losing data confidentiality. Some examples of such systems include e-Vault [33] and PASIS [53].

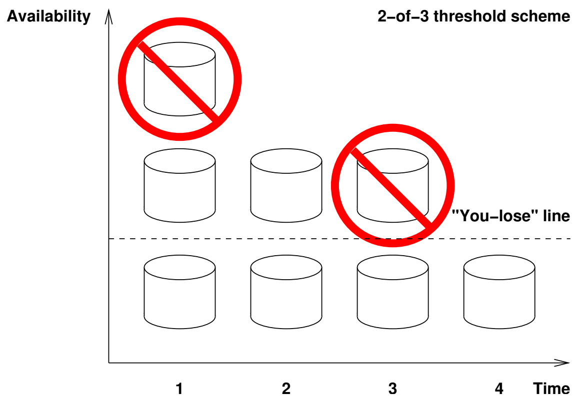

Figure 1.1: A storage system that uses a 2-of-3 threshold sharing scheme without recovery. Availability is plotted informally on the y axis, and time is plotted on the x axis. Suppose that the system suffers a server failure at time 1. Data remains available through time 2 because enough servers (*i.e.*, 2) remain to serve shares. However, suppose that the system suffers another failure at time 3. From that time forward, an insufficient number of servers remain.

A survivable storage system requires mechanisms to recover from component failures, as a threshold scheme by itself is insufficient for the long term. Consider the graph in Figure 1.1 of a three-component system that uses a 2-of-3 scheme to store data. If one assumes that all components will eventually fail, then the system will reach a point when only one non-failed component remains. By then, the end-user's data will be lost. To recover from failures in threshold sharing scheme-based systems, researchers have proposed *proactive secret sharing* (PSS) schemes [19, 20, 21, 31, 32, 44, 56, 55]. PSS schemes enable systems to replace lost shares and render compromised shares useless.

Recovery mechanisms for survivable storage systems must be *decentralized*. A naïve approach to recovery would be for a system to use a *recovery server* to reconstruct all of the stored data and redistribute new shares. An obvious shortcoming with this centralized approach is that it introduces a single point of failure in the system. If the recovery server itself crashes, recovery becomes impossible. Worse, unauthorized users who compromise the server immediately gain the ability to obtain all stored data.

This dissertation research contributes the first recovery mechanism that enables a system to adjust m and n, even if some of its components are controlled by unauthorized users. The limitation with mechanisms that only replace lost shares (such as PSS schemes) is that the assumptions behind the selection of m and n may prove to be invalid over time: components may be more crash-prone, or unauthorized users may be more aggressive in their attacks. The ability to change m and n enables a system to survive more crash-prone components, by increasing m, or to defend against more aggressive unauthorized users, by increasing n.

1.1 Thesis statement

The thesis statement of this dissertation is:

We can create a survivable storage system that:

- **Recovers from component failures,**
- **Adapts to changing requirements, and**
- **Accomplishes these goals in a decentralized manner.**

To support the thesis, I present the design and experimental performance analysis of a *verifiable secret redistribution* (VSR) protocol for Shamir's threshold sharing scheme [49]. The VSR protocol redistributes shares of data from old to new, possibly disjoint, sets of servers, such that new shares generated by redistribution cannot be combined with old shares to reconstruct the original data. The protocol is decentralized, and does not require intermediate reconstruction of the data; thus, one does not create a central point of failure or risk the exposure of the data during protocol execution. The protocol incorporates a verification capability that enables new servers to confirm that their shares can be used to reconstruct the original data.

The rest of this dissertation is organized as follows. In Chapter 2, I present an abstract model of a survivable storage system, and a model of a mobile adversary who subverts servers in the system and causes them to fail. I also postulate the design requirements for a recovery protocol in the context of a mobile adversary. In Chapter 3, I present the design of the VSR protocol, which can be used by a storage system to counteract the adversary discussed in Chapter 2. I prove that the shares held by servers after protocol execution can be used to reconstruct the original secret, and demonstrate that it satisfies all of the design requirements for a recovery protocol. In Chapter 4, I discuss the design and implementation of an experimental storage system called *Hathor*. The primary purpose of Hathor is to provide a platform on which to evaluate the end-to-end cost of storing, redistributing, and retrieving data using a variety of data distribution schemes (including Shamir's scheme). In Chapter 5, I present and analyze the results of experiments done to measure the raw computational performance of the VSR protocol, as well as the results of experiments done to measure the end-to-end cost of storing, redistributing, and retrieving data with Hathor. The results demonstrate that although the VSR protocol is computationally expensive, the cost can be offset through careful selection of the data distribution scheme. In Chapter 6, I survey the related work on survivable storage systems and recovery protocols for threshold sharing schemes. Finally, in Chapter 7, I end with a discussion of conclusions, research results, and directions of future work.

Chapter 2

System model

Lay not up for yourselves treasures upon earth, where moth and rust doth corrupt, and where thieves break through and steal.

— Matthew 6:19 (KJV)

In this chapter, I present an abstract model of a survivable storage system consisting of clients, servers, and a communication network. I also present a model of how an *adversary* might subvert the servers in such a system, and discuss how the subverted servers could attempt to disrupt system operation. I then discuss how the system can use a recovery protocol to counteract the actions of the adversary, and identify the design requirements for the protocol.

2.1 Abstract storage system model

The abstract storage system model consists of clients, servers, and a communications network. Figure 2.1 shows clients, servers, and the network channels that connect clients and servers to each other. In this section, I present a high-level description of the system components, and the assumptions I make about their behavior.

Clients are hosts that store and retrieve data with m-of-n distribution schemes. Clients are always *correct*: given data d and scheme s, a client applies s to d according to the specification of s. Servers are hosts that store pieces of data on stable storage. Servers are usually correct: given a piece p of data, a server saves p on stable storage when it receives p from a client, and returns p when requested by the client. However, a server may sometimes be *faulty*, exhibiting Byzantine behavior: given p,

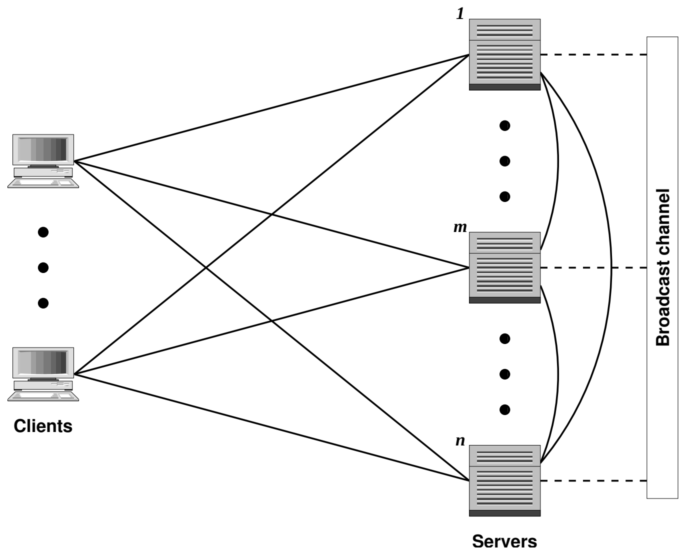

Figure 2.1: Abstract model of a storage system with n servers. The solid lines show point-to-point connections between components: clients are connected to servers, and a server is connected to all other servers. The dashed lines show connections of servers to a broadcast channel. Dots indicate elided clients or servers.

a faulty server may refuse to save p on stable storage, or save a corrupted piece p' instead of p, or refuse to return p. I assume that correct servers make forward progress; in particular, a server is deemed faulty if it does not send messages when expected to in a timely manner.

I assume that underlying authentication and permission mechanisms exist for a server to confirm that a client has the right to store a piece at the server. I also assume for now that the membership of the set of servers is constant, though I will relax this assumption later.

The network in the system provides point-to-point channels from a client to all servers, and from a server to all other servers. The channels deliver messages reliably in first-in, first-out (FIFO) order for each pair of hosts: if a host A sends a message M to host B, B is guaranteed to receive M, and guaranteed to receive M before any other message M' that A sends after sending M. The channels are authenticated: if A sends a message to B, no host E can masquerade as either A or B. Lastly, the channels are private: if A sends a message to B, its contents are known only to A and B.

The network also provides a broadcast channel that connects all servers. The channel delivers messages from a server reliably in FIFO order: if a server broadcasts a message M followed by M', all other servers receive M followed by M'. However, the channel does not enforce an ordering between messages sent by different servers: if server A broadcasts M_A, and server B broadcasts M_B, server C may receive M_A followed by M_B or M_B followed by M_A, and server D may receive

M_A and M_B in an order different from C. The channel is authenticated: no server E, regardless of whether it is correct or faulty, can masquerade as some other server (*i.e.*, E cannot forge signatures on messages).

2.2 The mobile adversary and its effect on servers

An *adversary* is an external entity that attempts to subvert servers. Once subverted, a server is controlled by the adversary, and may behave in ways that deviate from its specification. In this section, I present a temporal model of how an adversary may subvert target servers.

Conceptually, the adversary is an external host that is connected by point-to-point channels to all servers, while also being connected to the broadcast channel. The adversary is not connected to the clients, and in any case cannot subvert the clients (consistent with the assumption of correct clients). The adversary cannot eavesdrop on the point-to-point channels, or inject messages into those channels that purport to be from other hosts (consistent with the assumptions about the point-to-point channels). The adversary can see all messages sent over the broadcast channel, and can also send messages over the channel or inject messages via servers under its control. The adversary causes subverted servers to become faulty, and exhibit the Byzantine behavior discussed in Section 2.1. In turn, a faulty server may reveal its memory contents (*e.g.*, stored pieces of data) to the adversary.

To reason about the temporal behavior of the adversary, I adopt the *mobile adversary* model proposed by Ostrovsky and Yung [39] and refined by Herzberg *et al.* [32]. In the model, time is divided into *epochs* in which the adversary may subvert a limited number of servers. The limit is specified implicitly by the data distribution scheme used in the system, *e.g.*, for an m-of-n scheme, the adversary may control at most $m-1$ servers per epoch. An *update phase* separate consecutive epochs. At the start of the update phase, the system tries to remove all subverted servers from under the control of the adversary, though it might succeed in removing only some or none. A formerly subverted server may have corrupted memory contents. After the end of the update phase, the adversary may again subvert servers, up to the limit specified by the distribution scheme. The adversary is constrained from re-subverting servers during the update phase, which is reasonable provided that an update phase is short relative an epoch. Ostrovsky and Yung assume that the system has some external mechanism for the detection and repair of subverted servers.

One can visualize the mobile adversary as having a fixed number of "pebbles" that it may place on servers during each epoch. A server is covered by at most one pebble. At the start of the update phase, the system tries to remove all pebbles and gives them back to the adversary, though it might

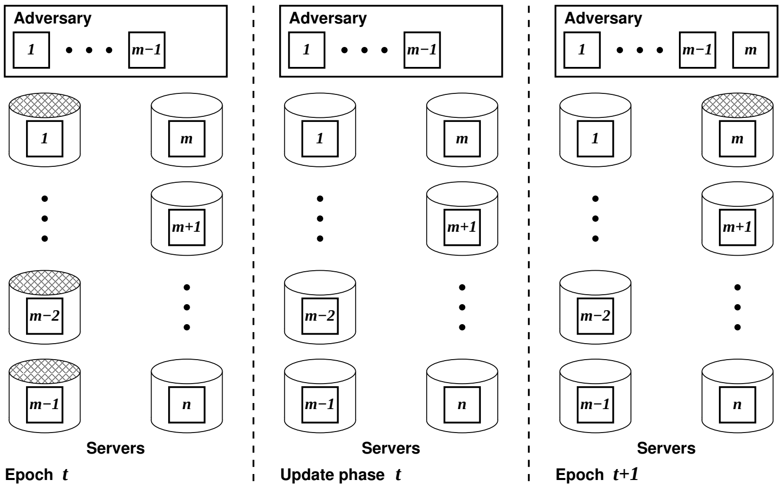

Figure 2.2: A storage system that uses an m-of-n data distribution scheme, in the presence of a mobile adversary. The memory contents (*i.e.*, pieces of data) of the adversary and servers are shown. The adversary may only control at most $m-1$ servers per epoch. Crosshatched servers are controlled by the adversary. In epoch t, the adversary subverts servers 1 through $m-1$ and obtains their pieces. During the update phase, the system removes all servers from under the control of the adversary. In epoch $t+1$, the adversary subverts server m and obtains its piece. The adversary can then reconstruct the original data.

succeed in removing only some or none. At the start of the next epoch, the adversary may again place pebbles on servers. A server that is covered by a pebble is controlled by the adversary. A covered server that is later uncovered is no longer controlled by the adversary.

Even though a mobile adversary can only subvert a limited number of servers in each epoch, it can eventually subvert every server over multiple epochs. For example, consider the system in Figure 2.2 that uses an m-of-n scheme. The adversary may control at most $m-1$ servers per epoch. Initially, in epoch t, the adversary subverts servers 1 through $m-1$, and obtains the pieces held by those servers. During the update phase, the system removes all of the servers from under the control of the adversary. In epoch $t+1$, the adversary subverts server m, and obtains the piece held by m; thus, the adversary may now reconstruct the original data.

To counteract the adversary, the system requires a recovery protocol to execute after it has tried to remove subverted servers from under the control of the adversary. I postulate the following protocol design requirements in the context of the mobile adversary model:

- It must generate new pieces for the next epoch such that they can be used to reconstruct the secret, and such that they cannot be combined with pieces from the current epoch to reconstruct the secret.

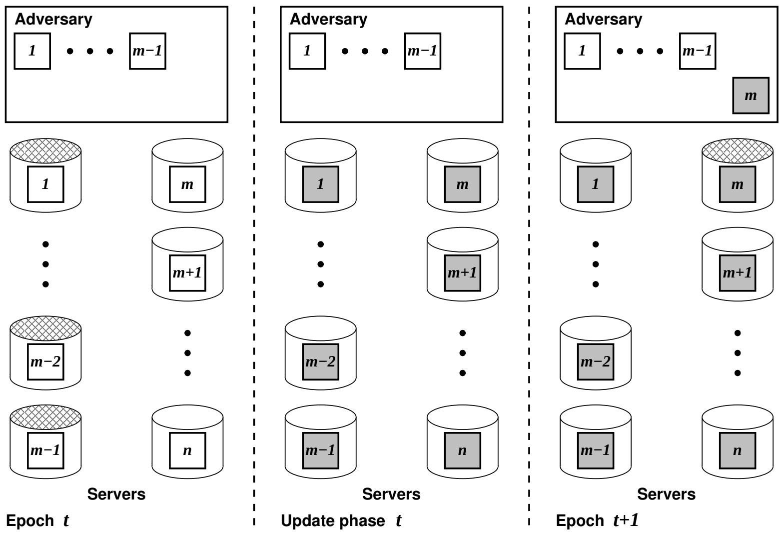

Figure 2.3: A storage system with a recovery protocol in the presence of a mobile adversary. The adversary may only control at most $m-1$ servers per epoch. Crosshatched servers are controlled by the adversary. The system executes a recovery protocol during the update phase to generate new (shaded) pieces for correct servers. New pieces cannot be combined with current pieces to reconstruct the original data. In epoch t, the adversary subverts servers 1 through $m-1$ and obtains their pieces. During the update phase, the system removes all servers from under the control of the adversary. In epoch $t+1$, the adversary subverts server m and obtains its piece. However, the adversary does not obtain enough pieces (current or new) to reconstruct the original data.

- It must include mechanisms to prevent the adversary from corrupting protocol execution, because the adversary may still control some servers during the update phase.
- It must erase the pieces for the current epoch from server memories, to prevent the adversary from ever obtaining any other current pieces it needs.

With a recovery protocol that satisfies these design requirements, the system can prevent a mobile adversary from ever obtaining enough pieces to reconstruct the original data; I will prove this point in Section 3.3. For example, consider the system in Figure 2.3 that uses an m-of-n scheme, and contrast it with the system in Figure 2.2. As before, the adversary may control at most $m-1$ servers per epoch. Initially, in epoch t, the adversary subverts servers 1 through $m-1$, and obtains the pieces held by those servers. During the update phase, the system removes all of the servers from under the control of the adversary, and executes the recovery protocol. In epoch $t+1$, the adversary subverts server m, and obtains the piece held by m; however, this piece cannot be combined with the pieces from t to reconstruct the data.

2.3 The dynamic membership model

Ostrovsky and Yung assume a *static membership model* of the system in their mobile adversary model [39]. In their model, the number of servers, the membership of the set of servers, and the threshold parameter of the underlying data distribution scheme are all fixed throughout the duration of system operation. In practice, however, one might wish for the system to exclude subverted servers from the system, replacing them with new servers. Also, one might wish to increase the number of servers in the system (and the threshold parameter of the underlying distribution scheme), in order to increase the number of servers the adversary must subvert in order to reconstruct data.

I adopt a *dynamic membership model* of a system by extending the mobile adversary model to allow servers to join and leave the system. As in the original model, I divide time into epochs during which an adversary may subvert a limited number of servers. Consecutive epochs are separated by an update phase. At the start of the update phase, the system tries to remove all subverted servers from under the control of the adversary (though it might succeed in removing only some or none), and admits new servers. It then executes the recovery protocol before allowing old servers to leave the system. After the end of the update phase, the adversary may again subvert servers. The adversary is constrained from subverting servers during the update phase, and from corrupting the state of underlying membership protocols that manage the set of servers. An old server, once it leaves the system, is treated like a new server if it tries to rejoin the system.

The dynamic membership model impacts the design of mechanisms put in place to prevent an adversary from corrupting recovery protocol execution (the second requirement in Section 2.2). The set of servers in the next epoch may be completely disjoint from the set in the current epoch, thus none of the new servers will have any information about the original data, (*e.g.*, none of them will have ever stored pieces of the data). This lack of information rules out simple adaptations of recovery protocols designed around static membership models, in which participant servers perform verification computations (to prevent an adversary from corrupting execution) in the current update phase that require information that they have received in the previous update phase.

I impose a new requirement on the recovery protocol to accommodate dynamic membership:

- It must enable the system to change the threshold parameter of the underlying data distribution scheme.

I motivate the new requirement with the following example. Consider a system of n servers that uses an n-of-n scheme. Suppose that the set of servers changes such that there are n' servers, where $n' < n$. The system must change the parameters of the underlying distribution scheme so that a lower number of pieces can be used to reconstruct the data.

Because the threshold parameter may change, I need to add a new pair of assumptions to the adversary model. Suppose the system uses an m-of-n scheme in the current epoch, and will use an m'-of-n' scheme in the next epoch. With these parameters, the adversary can control at most $m-1$ servers in the current epoch, and at most $m'-1$ servers in the next epoch. Also, at the start of the update phase, I assume that the system removes subverted servers from under the control of the adversary such that, during the update phase, at most $m-1$ servers from the current epoch and at most $m'-1$ servers from the next epoch are subverted. Previously, in a system that used an m-of-n scheme, the adversary could control at most $m-1$ servers per epoch; I assumed that the system would try to remove all subverted servers from under the control of the adversary at the start of the update phase, though it might succeed in removing only some or none.

2.4 Summary

I have presented the abstract storage system model that I will use throughout the remainder of this dissertation. Distinct from previous work, the set of servers in the system has a dynamic membership: the number of servers, the membership of the set, and the threshold parameter of the underlying data distribution scheme may change during the lifetime of the system. I have also discussed the design requirements for a recovery protocol that the system can use to counteract an adversary in the context of a dynamic membership model.

Chapter 3

Verifiable Secret Redistribution

Trust, but verify.

— *Russian proverb*

In this chapter, I present the *verifiable secret redistribution* (VSR) protocol for threshold sharing schemes. The VSR protocol is designed to counteract a mobile adversary in a system of servers with dynamic membership, and ensure that the servers have valid shares of data after redistribution. The protocol executes in the update phase between epochs, after the system has removed some (perhaps none) of the servers from under the control of the adversary. Any shares of data obtained by the adversary prior to protocol execution are rendered useless after successful execution, provided that the adversary had only obtained a sub-threshold number of shares. Moreover, the adversary cannot combine shares obtained prior to protocol execution with shares obtained after execution to reconstruct the data.

In the presentation of the VSR protocol, I employ the terminology that is generally used in the discussion of threshold sharing schemes. Thus, in this chapter I refer to data as *secrets*, clients as *dealers*, and servers as *shareholders*.

The rest of this chapter is organized as follows. In Section 3.1, I outline the cryptographic protocols that are the building blocks for the VSR protocol. In Section 3.2, I present the VSR protocol, and prove that shares held by shareholders after protocol execution can be used to reconstruct the original secret. In Section 3.3, I show that the VSR protocol fulfills all of the design requirements discussed in Chapter 2, and demonstrate how it counteracts a mobile adversary.

3.1 Cryptographic building blocks

In this section, I outline the building blocks of the VSR protocol: Shamir's threshold sharing scheme [49], Desmedt and Jajodia's secret redistribution protocol [16], and Feldman's VSS scheme [17].

3.1.1 Shamir's threshold sharing scheme

Shamir presents a scheme for distributing n shares of a secret to n shareholders, such that the shares of any subset of m unique shareholders can be used to reconstruct the secret [49]. Secrets k are in the finite field of integers $\mathbb{Z}_p$ (where p is prime and $p > n$). Shareholders i are in the set of participants $\mathcal{P}$ ($|\mathcal{P}| = n$). Shares s_i of i are also in the set $\mathbb{Z}_p$. Each subset of m unique shareholders forms an *authorized subset* $\mathcal{B}$; all authorized subsets are in the *access structure* $\Gamma_{\mathcal{P}}^{(m,n)}$.

To distribute k to $i \in \mathcal{P}$, a dealer selects a random $m-1$ degree polynomial $a(x)$ with constant term equal to k and random coefficients a_1 ... $a_{m-1} \in \mathbb{Z}_p$, and uses $a(x)$ to generate s_i for each i:

$$s_i = k + a_1 i + \ldots + a_{m-1} i^{m-1} \bmod p \tag{3.1}$$

To reconstruct k, the dealer retrieves m shares s_i of $i \in \mathcal{B}$, and uses Lagrange interpolation to recover the constant term of $a(x)$, *i.e.*, k:

$$k = \sum_{i \in \mathcal{B}} b_i s_i \bmod p \quad \text{where} \quad b_i = \prod_{j \in \mathcal{B}, j \neq i} \frac{j}{(j-i)} \bmod p \tag{3.2}$$

For the rest of the chapter, I omit the modulus operator to simply the notation.

3.1.2 Desmedt and Jajodia's secret redistribution protocol

Desmedt and Jajodia present a protocol for the redistribution of shares of secrets for threshold sharing schemes [16] that does not require the intermediate reconstruction of the secret. I summarize a specialized version of their protocol for use with Shamir's threshold sharing scheme [49], as shown in Figure 3.1. To redistribute a secret k from the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$, one selects an authorized subset $\mathcal{B} \in \Gamma_{\mathcal{P}}^{(m,n)}$. Each shareholder $i \in \mathcal{B}$ uses Shamir's scheme to distribute *subshares* $\hat{s}_{ij}$ of its share s_i to each shareholder $j \in \mathcal{P}'$:

$$\hat{s}_{ij} = s_i + a'_{i1'} j + \ldots + a'_{i(m'-1)} j^{m'-1} \tag{3.3}$$

Desmedt and Jajodia's Secret Redistribution protocol for Shamir's scheme
To redistribute a secret $k \in \mathbb{Z}_p$ from the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$ using the authorized subset $\mathcal{B} \in \Gamma_{\mathcal{P}}^{(m,n)}$:

1. For each $i \in \mathcal{B}$, use the random polynomial $a'_i(j) = s_i + a'_{i1} j + \ldots + a'_{i(m'-1)} j^{m'-1}$ to compute the subshares $\hat{s}_{ij}$ of s_i, and send $\hat{s}_{ij}$ to the corresponding $j \in \mathcal{P}'$.
2. For each $j \in \mathcal{P}'$, generate a new share s'_j by Lagrange interpolation:

$$s'_j = \sum_{i \in \mathcal{B}} b_i \hat{s}_{ij} \quad \text{where} \quad b_i = \prod_{x \in \mathcal{B}, x \neq i} \frac{x}{(x-i)}$$

   b_i are constant for each $i \in \mathcal{B}$, are independent of the choice of $a'_i(x)$, and may be precomputed.

Figure 3.1: Protocol for the redistribution of shares of a secret from the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$ [16] with Shamir's threshold sharing scheme [49].

Each j then generates a new share s'_j by Lagrange interpolation:

$$s'_j = \sum_{i \in \mathcal{B}} b_i \hat{s}_{ij} \quad \text{where} \quad b_i = \prod_{x \in \mathcal{B}, x \neq i} \frac{x}{(x-i)} \tag{3.4}$$

One can redistribute shares of k an arbitrary number of times prior to the reconstruction of k.

To reconstruct k after redistribution, one retrieves m' shares s'_j of $j \in \mathcal{B}'$, and uses Lagrange interpolation:

$$k = \sum_{j \in \mathcal{B}'} b'_j s_j \quad \text{where} \quad b'_j = \prod_{x \in \mathcal{B}', x \neq j} \frac{x}{(x-j)} \tag{3.5}$$

3.1.3 Feldman's VSS scheme

Feldman presents a scheme for verifiable secret sharing (VSS) [17] that shareholders of a secret can use to verify that their shares are *valid*; *i.e.*, the shares of any authorized subset of shareholders can be used to reconstruct the original secret.

To use Feldman's scheme, assume that there exists a homomorphic *witness function* $W(x)$ of the form

Feldman's Verifiable Secret Sharing scheme for Shamir's scheme
To distribute a secret $k \in \mathbb{Z}_p$ to the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$:

1. Use the random polynomial $a(i) = k + a_1 i + \ldots + a_{m-1} i^{m-1}$ to compute the shares s_i of k, and send s_i to the corresponding $i \in \mathcal{P}$ over private channels.
2. Use a witness function $W(x)$ to compute $W(k), W(a_1) \ldots W(a_{m-1})$, and send them to all $i \in \mathcal{P}$ over the broadcast channel.
3. For each $i \in \mathcal{P}$, verify that:

$$W(s_i) \equiv W(k) \oplus (W(a_1) \otimes i) \oplus \ldots \oplus (W(a_1) \otimes i^{m-1})$$

If the condition holds, i broadcasts a "commit" message. Otherwise, i broadcasts an "abort" message.

Figure 3.2: Feldman's VSS scheme [17] for Shamir's threshold sharing scheme [49].

$$W(a+b) = W(a) \oplus W(b) \tag{3.6}$$

$$W(ab) = W(a) \otimes b \tag{3.7}$$

for which inversion is intractable: that is, given $W(x)$ it is intractable to compute x. The witness functions allows one to prove knowledge of some value without revealing the value. The $\oplus$ and $\otimes$ operations are the homomorphic equivalents of addition and multiplication in the finite field of integers.

The VSS scheme works as follows. The dealer uses Shamir's scheme with a random polynomial $a(x)$ to distribute the secret $k \in \mathbb{Z}_p$ to the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$. In addition to sending the shares s_i to shareholders $i \in \mathcal{P}$, the dealer broadcasts *witnesses* of k and the coefficients a_1 ... a_{m-1} of $a(x)$, *i.e.*, $W(k)$ and $W(a_1)$ ... $W(a_{m-1})$. Each i may then verify that s_i is a valid share of k using

$$W(s_i) \equiv W(k) \oplus (W(a_1) \otimes i) \oplus \ldots \oplus (W(a_1) \otimes i^{m-1}) \tag{3.8}$$

which is the result of applying $W(x)$ to $a(x)$ (Equation (3.1)). Because the inversion of $W(x)$ is intractable, no one can learn k or a_1 ... a_{m-1} from the broadcast of the witnesses.

In this dissertation, I consider two candidate witness functions: exponentiation in a finite field and point multiplication on an elliptic curve.

Exponentiation

The finite field of integers $\mathbb{Z}_p$ (p prime) has a corresponding multiplicative ring $\mathbb{Z}_q^*$ (q prime; $q = pr+1$; r is a positive integer). Given $\mathbb{Z}_p$ and $\mathbb{Z}_q^*$, one can define a witness function

$$W(x) = g^x \quad \text{where} \quad x \in \mathbb{Z}_p \tag{3.9}$$

where integer $g \in \mathbb{Z}_q^*$ is a publicly-known *generator* of a sub-ring of $\mathbb{Z}_q^*$ of prime order. The intractability of the inversion of $W(x)$ is based on the *discrete logarithm problem* (DLP) for finite fields: given g and g^k, it is hard to compute k. The $\oplus$ operation is multiplication in $\mathbb{Z}_q^*$, and the $\otimes$ operation is exponentiation in $\mathbb{Z}_q^*$ (*cf.* Equation (3.6)):

$$\begin{aligned} W(a+b) &= g^a g^b && (3.10)\\ W(ab) &= (g^a)^b && (3.11) \end{aligned}$$

Point multiplication on an elliptic curve

An elliptic curve E over the finite field $\mathbb{F}_{p^m}$ (p prime; m is a positive integer) is denoted by $E(\mathbb{F}_{p^m})$, and is defined by the equation

$$\begin{aligned} y^2 = x^3 + ax + b \quad (p \geq 3) && (3.12)\\ y^2 + xy = x^3 + ax + b \quad (p = 2) && (3.13) \end{aligned}$$

where the coordinate points $x, y \in \mathbb{F}_{p^m}$. The points on the curve form a group comprising $|E(\mathbb{F}_{p^m})|$ points, with a rule that defines the addition of points P and Q and another rule that defines the multiplication of a point P by a scalar integer k. The group contains an additive identity called the *point at infinity* O (that is, a point P added to the point at infinity yields P). Blake, Seroussi, and Smart present a more comprehensive discussion of elliptic curves in cryptography [6].

Given an elliptic curve, one can define a witness function

$$W(x) = [x]G \quad \text{where} \quad 1 < x < r, [r]G = O \tag{3.14}$$

where $G \in E(\mathbb{F}_{p^m})$ is a publicly-known point (*cf.*, g for exponentiation) that generates a prime order subgroup of $E(\mathbb{F}_{p^m})$ of order $r < |E(\mathbb{F}_{p^m})|$. The notation $[x]G$ distinguishes the scalar variable x from the multi-dimensional coordinate G. The intractability of the inversion of $W(x)$ is based on the analog to the DLP for elliptic curves: given P and $[k]P$, it is hard to compute k. The $\oplus$ operation is point addition, and the $\otimes$ operation is scalar multiplication (*cf.* Equation (3.6)):

$$W(a+b) = [a]G + [b]G \tag{3.15}$$

$$W(ab) = [b]([a]G) \tag{3.16}$$

3.2 The VSR protocol

Here, I present the verifiable secret redistribution protocol for secrets distributed with Shamir's threshold sharing scheme [49]. The protocol takes as input shares of a secret distributed to an authorized subset $\mathcal{B}$ in the access structure $\Gamma^{(m,n)}_{\mathcal{P}}$, and outputs shares of the secret distributed to the access structure $\Gamma^{(m',n')}_{\mathcal{P}'}$. I assume that there exists a witness function $W(x)$ with the properties shown in Equation (3.6), and assume that inversion of $W(x)$ is intractable. The system model is the one presented in Section 2.1, in which the dealer is always correct, but in which some of the shareholders may be subverted by an adversary. I assume that there are at least m correct old shareholders, and that there are at most $m-1$ faulty old shareholders. I also assume that there are at least m' correct new shareholders, and that there are at most $m'-1$ faulty new shareholders. I assume that correct shareholders make forward progress; a shareholder is deemed faulty if it does not send protocol messages in a timely manner. The dealer and shareholders are fully connected to each other by private channels, and shareholders are also connected to each other by a broadcast channel.

The initial distribution of a secret (INITIAL in Figure 3.3) is with Shamir's threshold scheme [49]. The dealer of secret k distributes shares s_i to each shareholder $i \in \mathcal{P}$, using the random polynomial $a(i)$ (step 1 of INITIAL). The dealer also sends the witness $W(k)$ to each i (step 2). Each i receives the same value for $W(k)$, consistent with the assumption that the dealer is correct.

Redistribution of the secret (REDIST in Figure 3.3) is similar to Desmedt and Jajodia's protocol [16]. Each i in an authorized subset $\mathcal{B} \in \Gamma^{(m,n)}_{\mathcal{P}}$ uses Shamir's scheme (with the random polynomial $a'_i(j)$) to distribute subshares $\hat{s}_{ij} \in \mathbb{Z}_p$ of its share s_i to shareholders $j \in \mathcal{P}'$ (step 1 of REDIST). Each j receives $\hat{s}_{ij}$ from each i, and generates a new share s'_j (Equation (3.4), which is step 4). One can redistribute shares of k an arbitrary number of times prior to reconstruction of k.

Verifiable Secret Redistribution protocol for Shamir's sharing scheme

INITIAL

To distribute a secret $k \in \mathbb{Z}_p$ to the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$:

1. Use the random polynomial $a(i) = k + a_1 i + \ldots + a_{m-1} i^{m-1}$ to compute the shares s_i of k, and send s_i to the corresponding $i \in \mathcal{P}$ over private channels.
2. Use witness function $W(x)$ to compute $W(k)$, and send it to all $i \in \mathcal{P}$ over private channels.

REDIST

To redistribute $k \in \mathbb{Z}_p$ from the authorized subset $\mathcal{B}$ in the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$:

1. For each $i \in \mathcal{B}$, use the random polynomial $a'_i(j) = s_i + a'_{i1} j + \ldots + a'_{i(m'-1)} j^{m'-1}$ to compute the subshares $\hat{s}_{ij}$ of s_i, and send $\hat{s}_{ij}$ to the corresponding $j \in \mathcal{P}'$ over private channels. An i that has a null share sends nothing. A j that does not receive a timely $\hat{s}_{ij}$ from i broadcasts an "abort" message.
2. For each $i \in \mathcal{B}$, use $W(x)$ to compute $W(s_i), W(a'_{i1}) \ldots W(a'_{i(m'-1)})$, and send them and $W(k)$ to all $j \in \mathcal{P}'$ over the broadcast channel. An i that has a null share sends nothing. A j that does not receive a timely broadcast from i broadcasts an "abort" message.
3. For each $j \in \mathcal{P}'$, verify that:

$$\forall i \in \mathcal{B} : W(\hat{s}_{ij}) \equiv W(s_i) \oplus (W(a'_{i1}) \otimes j) \oplus \ldots \oplus (W(a'_{i(m'-1)}) \otimes j^{m'-1})$$

   and:

$$W(k) \equiv \bigoplus_{i \in \mathcal{B}} W(s_i) \otimes b_i \quad \text{where} \quad b_i = \prod_{l \in \mathcal{B}, l \neq i} \frac{l}{(l-i)}$$

   If the conditions hold, j broadcasts a "commit" message. Otherwise, j broadcasts an "abort" message. A j that does not send a timely "commit" message is assumed to have implicitly sent an "abort" message.
4. If at least $2m' - 1$ $j \in \mathcal{P}'$ broadcast "commit" messages, each j generates a new share s'_j:

$$s'_j = \sum_{i \in \mathcal{B}} b_i \hat{s}_{ij} \quad \text{where} \quad b_i = \prod_{l \in \mathcal{B}, l \neq i} \frac{l}{(l-i)}$$

   and stores s'_j and $W(k)$; all $i \in \mathcal{P}$ then erase their shares. A j that has received fewer than m subshares generates a null share.
5. If at least m' $j \in \mathcal{P}'$ broadcast "abort" messages, all $i \in \mathcal{B}$ and all $j \in \mathcal{P}'$ abort the protocol.

Figure 3.3: Protocol for the verifiable redistribution of shares of a secret from the authorized subset $\mathcal{B}$ in the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$ with Shamir's threshold sharing scheme [49].

For the new shareholders to have a valid sharing of the secret after redistribution, two conditions, called SHARES-VALID and SUBSHARES-VALID, must hold:

SHARES-VALID:

$k = \sum_{i \in \mathcal{B}} b_i s_i$

SUBSHARES-VALID:

$\exists \mathcal{B}' \in \Gamma_{\mathcal{P}'}^{(m',n')} : \forall i \in \mathcal{B} : s_i = \sum_{j \in \mathcal{B}'} b'_j \hat{s}_{ij}$

The VSR protocol must ensure that correct new shareholders have a valid sharing after protocol execution, *i.e.*, that there exists an authorized subset with shares that can be used to reconstruct the original secret. Thus, I define a NEW-SHARES-VALID condition, which holds if an authorized subset of new shareholders have valid shares of the secret. I prove in Section 3.2.4 that NEW--SHARES-VALID holds if SHARES-VALID and SUBSHARES-VALID hold. The definition of NEW-SHARES-VALID is similar to Equation (3.2):

NEW-SHARES-VALID:

$\exists \mathcal{B}' \in \Gamma_{\mathcal{P}'}^{(m',n')} : k = \sum_{j \in \mathcal{B}'} b'_j s'_j$

The definition of NEW-SHARES-VALID may seem counterintuitive, as one may expect *any* subset of m' new shareholders to have shares than can be used to reconstruct k. However, observe that some new shareholders may not have valid shares. First, faulty new shareholders may not have valid shares. Second, when $2m' - 1$ shareholders broadcast a "commit" message (step 5 of REDIST), there may exist new shareholders that have received fewer than m subshares, and thus cannot generate new shares; such shareholders must store a *null* share. As I will prove in Section 3.2.4, there will still exist at least m' new shareholders that can generate new shares.

I use Feldman's VSS scheme [17] to verify that SUBSHARES-VALID holds. The distribution of $\hat{s}_{ij}$ from s_i (step 1 of REDIST) is just an application of Shamir's scheme. Thus, each $i \in \mathcal{B}$ broadcasts witnesses of its share and the coefficients of $a'_i(j)$, $W(s_i)$ and $W(a_{i1})$ ... $W(a_{i(m-1)})$, which each j uses to verify the validity of $\hat{s}_{ij}$ (step 2).

The key insight embodied in the VSR protocol is that the naïve extension of Desmedt and Jajodia's protocol with Feldman's scheme does not in itself allow the new shareholders to verify that NEW-SHARES-VALID holds. The difficulty arises because Feldman's scheme only verifies that SUBSHARES-VALID holds, which in the absence of SHARES-VALID is insufficient to verify that NEW--SHARES-VALID holds. Although Desmedt and Jajodia observe that the linear properties of their protocol and the properties of $W(x)$ ensure that each j generates valid shares [16], they implicitly assume that each $i \in \mathcal{B}$ distributes subshares of valid s_i. The VSS scheme only enables i to prove that it distributed valid $\hat{s}_{ij}$ of some value. However, i may have distributed "subshares" of some random value instead of subshares of s_i. Thus, one requires a sub-protocol for i to prove that it distributed $\hat{s}_{ij}$ of s_i.

A similar flaw can be found in the proactive RSA scheme proposed by Frankel *et al.* [18]. Their protocol uses a poly-to-sum redistribution from a polynomial sharing scheme to an additive sharing scheme, and a sum-to-poly redistribution from the additive scheme back to a polynomial scheme. They suggest that changes in the threshold and number of shareholders can be accommodated in the poly-to-sum redistribution. Unfortunately, their verification checks hold only if one retains the same set of shareholders, because their "proper secret" check relies on a witness ($g^{s_i}L^2$ in their paper) computed from information distributed in the preceding execution of the protocol.

To allow the new shareholders to verify that SHARES-VALID holds, the old shareholders in the protocol broadcast a witness of the original secret k. Each $i \in \mathcal{B}$ stores $W(k)$ received during INITIAL and later broadcast it to all $j \in \mathcal{P}'$. Recall that each j receives $W(s_i)$ from each i to verify that SUBSHARES-VALID holds. Once j receives $W(k)$, it verifies that each s_i is a valid share of k:

$$W(k) = \bigoplus_{i \in \mathcal{B}} W(s_i) \otimes b_i \tag{3.17}$$

which is the result of applying $W(x)$ to Equation (3.2). Because inversion of $W(x)$ is intractable, no-one can learn k directly from the broadcast of $W(k)$.

3.2.1 Assumptions about faulty shareholders

During redistribution from the authorized subset $\mathcal{B}$ in access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$ with the VSR protocol, I assume that at least m shareholders in $\mathcal{P}$ are correct, and that at most $m-1$ shareholders in $\mathcal{P}$ are faulty. I also assume that at least m' shareholders in $\mathcal{P}'$ are correct, and that at most $m'-1$ shareholders in $\mathcal{P}'$ are faulty. I denote faulty shareholders and invalid values with over-bars. A correct shareholder $i \in \mathcal{P}$ distributes valid subshares $\hat{s}_{ij}$ of its share s_i to all shareholders $j \in \mathcal{P}'$ and broadcasts $W(k)$ corresponding to secret k (REDIST in Figure 3.3). A faulty shareholder $\bar{\imath} \in \mathcal{P}$ may distribute invalid subshares $\overline{\hat{s}_{\bar{\imath}j}}$ or broadcast $\overline{W(k)}$ not corresponding to k; of course, $\bar{\imath}$ may instead distribute valid subshares or valid witnesses.

One can derive a relationship between the old threshold scheme parameters m and n. If at least m old shareholders must be correct and at most $m-1$ old shareholders may be faulty, it is required that $m+m-1 \leq n$, or

$$m \leq \left\lfloor \frac{n+1}{2} \right\rfloor \tag{3.18}$$

One can also derive a relationship between the new threshold scheme parameters m' and n'. To avoid a race condition in which some new shareholders have received at least $2m'-1$ "commit" messages and fewer than m' "abort" messages, while others have received at least m' "abort" messages but fewer than $2m'-1$ "commit" messages, it is required that $(2m'-1)+(m'-1) \geq n'$. Also, to ensure that new shareholders broadcast at least $2m'-1$ "commit" messages, it is required that $2m'-1 \leq n'$. Hence,

$$\left\lceil \frac{n'+2}{3} \right\rceil \leq m' \leq \left\lfloor \frac{n'+1}{2} \right\rfloor \tag{3.19}$$

3.2.2 Detection of faulty old shareholders

The VSR protocol enables new shareholders to detect that some subset of the old shareholders are faulty. However, depending on the actions taken by faulty shareholders, the new shareholders may not be able to pinpoint the identity of the faulty shareholders. I consider two different scenarios for redistribution between access structures $\Gamma_{\mathcal{P}}^{(m,n)}$ and $\Gamma_{\mathcal{P}'}^{(m',n')}$, and show the circumstances in which the new shareholders can pinpoint a faulty old shareholder $\bar{\imath}$ in an authorized subset $\mathcal{B} \in \Gamma_{\mathcal{P}}^{(m,n)}$.

First, suppose that $\bar{\imath} \in \mathcal{B}$ broadcasts valid witnesses $W(k)$, $W(s_{\bar{\imath}})$ and $W(a_{\bar{\imath}1})$ ... $W(a_{\bar{\imath}(m-1)})$ (step 2 of REDIST in Figure 3.3). If $\bar{\imath}$ sends an invalid subshare $\overline{\hat{s}_{\bar{\imath}j}}$ to $j \in \mathcal{P}'$, j will find that the SUBSHARES-VALID condition does not hold. j can pinpoint $\bar{\imath}$, because only $\bar{\imath}$ can generate the information used to verify whether or not SUBSHARES-VALID holds (*i.e.*, $W(s_{\bar{\imath}})$, $W(a_{\bar{\imath}1})$ ... $W(a_{\bar{\imath}(m-1)})$, and $\hat{s}_{\bar{\imath}j}$). j will therefore broadcast an "abort" message (step 3 of REDIST).

On the other hand, suppose that $\bar{\imath}$ broadcasts an invalid witness $\overline{W(k)} \neq W(k)$ (or an invalid witness $\overline{W(s_{\bar{\imath}})}$), while the other shareholders $i \in \mathcal{B}$ broadcast the valid witness $W(k)$ (or valid witnesses $W(s_i)$). j will find that the SHARES-VALID condition does not hold. However, j cannot pinpoint $\bar{\imath}$, because j cannot distinguish between the case where $\bar{\imath}$ broadcasts an invalid witness, and the case where $\bar{\imath}$ is correct and the other shareholders in $\mathcal{B}$ have conspired to broadcast invalid witnesses.

Any randomly selected authorized subset $\mathcal{B}$ must contain at least one correct shareholder (consistent with the assumption that at most $m-1$ old shareholders are faulty). If the new shareholders find that one of the SHARES-VALID or SUBSHARES-VALID conditions does not hold, they can restart the redistribution protocol with another authorized subset until both conditions hold. For $\Gamma_{\mathcal{P}}^{(m,n)}$, the number of times the new shareholders must restart the redistribution protocol is bounded in the worst case by the number of sets of size m containing at least one faulty shareholder, given $m-1$ faulty shareholders:

$$\binom{n}{m} - \binom{n-m+1}{m} = \sum_{i=1}^{m-1} \binom{m-1}{i} \binom{n-m+1}{m-i} \tag{3.20}$$

The VSR protocol does not specify a restart mechanism. A system that incorporates the VSR protocol would need to implement a mechanism to detect "abort" messages (step 3 of REDIST in Figure 3.3), and restart the protocol with a different authorized subset in $\Gamma_{\mathcal{P}}^{(m,n)}$.

3.2.3 Computation cost

The computation cost of verification for each old shareholder in the VSR protocol (step 2 of REDIST in Figure 3.3) is $O(m')\, W(x)$ computations. Consider redistribution from the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$. Each old shareholder $i \in \mathcal{B}$ ($\mathcal{B} \in \Gamma_{\mathcal{P}}^{(m,n)}$) computes $W(s_i)$ for its share s_i, and $W(a_{ij})$ for each of the $m'-1$ coefficients a_{ij} in the subshare generation polynomial $a_i(x)$, for a total cost of $O(m')$.

The computation cost of verification for each new shareholder (step 3 of REDIST) is $O(m)\, W(x)$ computations, $O(mm')$ $\oplus$ operations, and $O(mm')$ $\otimes$ operations. Again, consider redistribution from the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$. Each new shareholder $j \in \mathcal{P}'$ computes $W(\hat{s}_{ij})$ to obtain a witness of the subshare $\hat{s}_{ij}$ from each i ($i \in \mathcal{B}$, $|\mathcal{B}| = m$), for a total cost of $O(m)$. Each j also performs $m'-1$ $\oplus$ operations ($\mathcal{B}' \in \Gamma_{\mathcal{P}'}$; $|\mathcal{B}'| = m'$) and $m'-1$ $\otimes$ operations for m old shareholders $i \in \mathcal{B}$ to verify that SUBSHARES-VALID holds (Equation (3.8)), for a total cost of $O(mm')$. Finally, each j also performs $m-1$ $\oplus$ operations and m $\otimes$ operations to verify that SHARES-VALID holds (Equation (3.17)), for a total cost of $O(m)$; I exclude the (small) cost of computing the powers of i because it is generally small compared to the cost of $W(x)$, $\oplus$, and $\otimes$ (Chapter 5).

3.2.4 Protocol correctness on termination

For the verifiable redistribution of shares of a secret k from an authorized subset $\mathcal{B}$ in the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$, I prove that if at least $2m'-1$ new shareholders broadcast a "commit" message, SHARES-VALID and SUBSHARES-VALID both hold. I then prove that SHARES-VALID and SUBSHARES-VALID are sufficient conditions for NEW-SHARES-VALID.

Lemma 1 *If at least* $2m'-1$ *shareholders in* $\mathcal{P}'$ *broadcast a "commit" message,* SUBSHARES-VALID *holds.*

PROOF: Assume that at least $2m'-1$ shareholders in $\mathcal{P}'$ broadcast a "commit" message. Also, recall the assumption that, at most, $m'-1$ shareholders in $\mathcal{P}'$ are faulty. I then need to prove that SUBSHARES-VALID holds.

Consider the shares s_i of old shareholders $i \in \mathcal{B}$, and the subshares $\hat{s}_{ij}$ that are distributed to new shareholders $j \in \mathcal{P}'$. At most, $m'-1$ "commit" messages originate from faulty new shareholders. Because at least $2m'-1$ new shareholders broadcast "commit" messages, at least m' messages must originate from correct shareholders that together constitute an authorized subset $\mathcal{B}' \in \Gamma_{\mathcal{P}'}^{(m',n')}$. Each $j \in \mathcal{B}'$ broadcasts a "commit" message only if Equation (3.8) holds for its subshare $\hat{s}_{ij}$ from each i (step 3 of REDIST in Figure 3.3), *i.e.*,

$$\forall j \in \mathcal{B}' : \forall i \in \mathcal{B} : W(\hat{s}_{ij}) \equiv W(s_i) \oplus (W(a'_{i1}) \otimes j) \oplus \ldots \oplus (W(a'_{i(m'-1)}) \otimes j^{m'-1})$$

which (from the homomorphic properties of the witness function $W(x)$) is equivalent to

$$\forall j \in \mathcal{B}' : \forall i \in \mathcal{B} : \hat{s}_{ij} = s_i + a'_{i1} j \ldots a'_{i(m'-1)} j^{m'-1}$$

The $\hat{s}_{ij}$ of j can be used to reconstruct s_i by Lagrange interpolation, *i.e.*,

$$\forall i \in \mathcal{B} : s_i = \sum_{j \in \mathcal{B}'} b'_j \hat{s}_{ij}$$

□

Lemma 2 *If at least m' shareholders in $\mathcal{P}'$ broadcast a "commit" message,* SHARES-VALID *holds.*

PROOF: Assume that at least m' shareholders in $\mathcal{P}'$ broadcast a "commit" message. Also, recall the assumption that, at most, $m'-1$ shareholders in $\mathcal{P}'$ are faulty. I then needs to prove that SHARES-VALID holds.

Consider the secret k, the shares s_i of old shareholders $i \in \mathcal{B}$, and the witnesses $W(s_i)$ of the shares and $W(k)$ of the secret. At most, $m'-1$ "commit" messages originate from faulty shareholders. Any remaining "commit" messages must originate from correct shareholders $j \in \mathcal{P}'$. Each j broadcasts a "commit" message only if Equation (3.17) holds (step 3 of REDIST in Figure 3.3), *i.e.*,

$$W(k) = \bigoplus_{i \in \mathcal{B}} W(s_i) \otimes b_i$$

which (from the homomorphic properties of the witness function $W(x)$) is equivalent to

$$k = \sum_{i \in \mathcal{B}} b_i s_i$$

Note that because the witnesses are sent by reliable broadcast, all new shareholders will see the same values. Thus, if SHARES-VALID holds at one correct shareholder, it will hold at all correct shareholders. □

Theorem 1 (VSR correctness on termination) *If* SHARES-VALID *and* SUBSHARES-VALID *hold,* NEW-SHARES-VALID *holds.*

PROOF: Assume that SHARES-VALID and SUBSHARES-VALID hold. One then needs to prove that NEW-SHARES-VALID holds.

The correctness proof is similar to that for Desmedt and Jajodia's secret redistribution protocol [16]. There exists $\mathcal{B}' \in \Gamma_{\mathcal{P}'}^{(m',n')}$ such that:

$$
\begin{aligned}
k &= \sum_{i \in \mathcal{B}} b_i s_i \qquad \text{(Lemma 2)} \\
&= \sum_{i \in \mathcal{B}} \left(b_i \sum_{j \in \mathcal{B}'} b'_j \hat{s}_{ij} \right) \qquad \text{(Lemma 1)} \\
&= \sum_{i \in \mathcal{B}} \sum_{j \in \mathcal{B}'} b_i b'_j \hat{s}_{ij} \qquad (x(y+z) = xy + xz) \\
&= \sum_{i \in \mathcal{B}} \sum_{j \in \mathcal{B}'} b'_j b_i \hat{s}_{ij} \qquad (xy = yx) \\
&= \sum_{j \in \mathcal{B}'} \sum_{i \in \mathcal{B}} b'_j b_i \hat{s}_{ij} \qquad (x + y = y + x) \\
&= \sum_{j \in \mathcal{B}'} \left(b'_j \sum_{i \in \mathcal{B}} b_i \hat{s}_{ij} \right) \qquad (xy + xz = x(y+z)) \\
&= \sum_{j \in \mathcal{B}'} b'_j s'_j \qquad \text{(Equation (3.4))}
\end{aligned}
$$

□

3.2.5 Protocol security

For the verifiable redistribution of shares of a secret k from an authorized subset $\mathcal{B}$ in the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$, I show that an adversary cannot reconstruct k from a combination of shares from the old and new sets of shareholders. In particular, I prove a lemma, Lemma 7, that an adversary cannot combine the shares of shareholders in the non-authorized subset $\overline{\mathcal{B}} \notin \Gamma_{\mathcal{P}}^{(m,n)}$ ($|\overline{\mathcal{B}}| < m$) and the shares of shareholders in the non-authorized subset $\overline{\mathcal{B}'} \notin \Gamma_{\mathcal{P}'}^{(m',n')}$ ($|\overline{\mathcal{B}'}| < m'$) to uniquely determine k. I have not been able to prove a second lemma, Lemma 8, that a computationally-bound adversary cannot use the shares of shareholders in $\overline{\mathcal{B}}$ with the witnesses $W(k), W(s_1), \ldots, W(s_m)$ to uniquely determine k. I do, however, provide a conjecture (and corresponding proof sketch) that would follow from Lemmas 7 and 8: a computationally-bound adversary cannot use the shares of shareholders in $\overline{\mathcal{B}}$ and $\overline{\mathcal{B}'}$, and the witnesses $W(k), W(s_1), \ldots, W(s_m)$ to uniquely determine k.

I require some lemmas presented by Beaumont [4] and Kostrikin [36] for systems of u linear equations in v unknowns of the form

$$\begin{aligned} m_{11}x_1 + m_{12}x_2 + \cdots + m_{1v}x_v &= b_1 \\ m_{21}x_1 + m_{22}x_2 + \cdots + m_{2v}x_v &= b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ m_{u1}x_1 + m_{u2}x_2 + \cdots + m_{uv}x_v &= b_u \end{aligned} \tag{3.21}$$

Let $\mathbf{M}$, $\mathbf{x}$, and $\mathbf{b}$ denote

$$\mathbf{M} = \begin{bmatrix} m_{11} & \cdots & m_{1v} \\ \vdots & \ddots & \vdots \\ m_{u1} & \cdots & m_{uv} \end{bmatrix} \quad , \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_v \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_u \end{bmatrix}$$

let $[\mathbf{M}|\mathbf{b}]$ denote the *augmented matrix*

$$[\mathbf{M}|\mathbf{b}] = \begin{bmatrix} m_{11} & \cdots & m_{1v} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ m_{u1} & \cdots & m_{uv} & b_u \end{bmatrix}$$

let $rank(\mathbf{M})$ denote the rank of $\mathbf{M}$ (number of linearly independent columns in $\mathbf{M}$), and let $det(\mathbf{M})$ denote the determinant of $\mathbf{M}$.

Lemma 3 $rank(\mathbf{M}) = rank(\mathbf{M}^T)$.

Lemma 4 (Kronecker-Capelli theorem) *Iff $rank(\mathbf{M}) = rank([\mathbf{M}|\mathbf{b}])$, Equation (3.21) has a solution for $\mathbf{x}$. Furthermore, if $rank(\mathbf{M}) < v$, Equation (3.21) has no unique solution for $\mathbf{x}$.*

Lemma 5 (Cramer's rule) *If $u = v$ and $det(\mathbf{M}) \neq 0$, Equation (3.21) has a unique solution for $\mathbf{x}$.*

Lemma 6 *For $u \times u$ matrix $\mathbf{A}$, $v \times v$ matrix $\mathbf{B}$, and $u \times v$ matrix $\mathbf{C}$,*

$$det\left(\begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}\right) = det(\mathbf{A})det(\mathbf{B})$$

PROOF: By Kostrikin [36]. □

Lemma 7 (VSR share security) *An adversary cannot combine the shares s_i of shareholders i in any non-authorized subset $\overline{\mathcal{B}} \notin \Gamma_{\mathcal{P}}^{(m,n)}$ ($|\overline{\mathcal{B}}| < m$) with the shares s'_j of shareholders j in any non-authorized subset $\overline{\mathcal{B}'} \notin \Gamma_{\mathcal{P}'}^{(m',n')}$ ($|\overline{\mathcal{B}'}| < m'$) to uniquely determine k.*

PROOF: Assume that there is a unique solution for k from the shares of shareholders in $\overline{\mathcal{B}}$ and $\overline{\mathcal{B}'}$. I will show that this assumption leads to a contradiction.

Suppose that $|\overline{\mathcal{B}}| = m-1$ and $|\overline{\mathcal{B}'}| = m'-1$, and suppose that the adversary has obtained s_i of $i \in \overline{\mathcal{B}}$ and s'_j of $j \in \overline{\mathcal{B}'}$. Without loss of generality, suppose that the shares are $s_1, \ldots, s_{m-1}, s'_1, \ldots, s'_{m'-1}$. Equation (3.1) can be used to construct the system of equations

$$\begin{bmatrix} 1 & 1 & \cdots & 1^{m-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & & \vdots \\ 1 & i & \cdots & i^{m-1} & \vdots & \ddots & \vdots \\ \vdots & \vdots & \cdots & \vdots & \vdots & & \vdots \\ 1 & (m-1) & \cdots & (m-1)^{m-1} & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 & 1 & \cdots & 1^{m'-1} \\ 1 & \vdots & & \vdots & \vdots & \cdots & \vdots \\ 1 & \vdots & \ddots & \vdots & j & \cdots & j^{m'-1} \\ 1 & \vdots & & \vdots & \vdots & \cdots & \vdots \\ 1 & 0 & \cdots & 0 & (m'-1) & \cdots & (m'-1)^{m'-1} \end{bmatrix} \begin{bmatrix} k \\ a_1 \\ \vdots \\ a_{m-1} \\ a'_1 \\ \vdots \\ a'_{m'-1} \end{bmatrix} = \begin{bmatrix} s_1 \\ \vdots \\ s_i \\ \vdots \\ s_{m-1} \\ s'_1 \\ \vdots \\ s'_j \\ \vdots \\ s'_{m'-1} \end{bmatrix} \tag{3.22}$$

Let $\mathbf{M}$ denote the left-hand matrix in Equation (3.22), $\mathbf{a}$ the coefficient vector k, a_1 ... $a'_{m'-1}$, and $\mathbf{s}$ the share vector. The maximum possible value for $rank(\mathbf{M})$ is the number of rows ($m+m'-2$, by Lemma 3), which is less than the number of values in $\mathbf{a}$ ($m+m'-1$). Also, $rank(\mathbf{M}) = rank([\mathbf{M}|\mathbf{s}])$ since $\mathbf{s}$ is a linear combination of the columns of $\mathbf{M}$ (by the method of share generation). Thus, there are no unique solutions for $\mathbf{a}$ in Equation (3.22) (by Lemma 4). One arrives at the same conclusion for any $\overline{\mathcal{B}'}' \notin \Gamma_{\mathcal{P}'}^{(m',n')}$ such that $|\overline{\mathcal{B}'}'| < m'-1$.

By assumption, there is a unique solution for k, thus Equation (3.22) can be re-written as

$$\begin{bmatrix} 1 & \cdots & 1^{m-1} & 0 & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & & \vdots \\ i & \cdots & i^{m-1} & \vdots & \ddots & \vdots \\ \vdots & \cdots & \vdots & \vdots & & \vdots \\ (m-1) & \cdots & (m-1)^{m-1} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 1^{m'-1} \\ \vdots & & \vdots & \vdots & \cdots & \vdots \\ \vdots & \ddots & \vdots & j & \cdots & j^{m'-1} \\ \vdots & & \vdots & \vdots & \cdots & \vdots \\ 0 & \cdots & 0 & (m'-1) & \cdots & (m'-1)^{m'-1} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_{m-1} \\ a'_1 \\ \vdots \\ a'_{m'-1} \end{bmatrix} = \begin{bmatrix} s_1 - k \\ \vdots \\ s_i - k \\ \vdots \\ s_{m-1} - k \\ s'_1 - k \\ \vdots \\ s'_j - k \\ \vdots \\ s'_{m'-1} - k \end{bmatrix} \tag{3.23}$$

Let $\mathbf{M_k}$ denote the left-hand matrix in Equation (3.23), and let $\mathbf{a_k}$ denote the coefficient vector a_1 ... $a'_{m'-1}$. Let $\mathbf{M_k^{UL}}$ and $\mathbf{M_k^{LR}}$ denote the upper-left and lower-right square sub-matrices of $\mathbf{M_k}$

$$\mathbf{M_k^{UL}} = \begin{bmatrix} 1 & \cdots & 1^{m-1} \\ \vdots & \ddots & \vdots \\ (m-1) & \cdots & (m-1)^{m-1} \end{bmatrix}$$

and

$$\mathbf{M_k^{LR}} = \begin{bmatrix} 1 & \cdots & 1^{m'-1} \\ \vdots & \ddots & \vdots \\ (m'-1) & \cdots & (m'-1)^{m'-1} \end{bmatrix}$$

One can express $det(\mathbf{M_k^{UL}})$ as

$$det(\mathbf{M_k^{UL}}) = 1 \cdots (m-1) \begin{vmatrix} 1 & \cdots & 1^{m-2} \\ \vdots & \ddots & \vdots \\ 1 & \cdots & (m-1)^{m-2} \end{vmatrix} = 1 \cdots (m-1) \prod_{1 \le i,j \le m-1; i > j} (i-j)$$

and observe immediately that $det(\mathbf{M_k^{UL}})$ and $det(\mathbf{M_k^{LR}})$ are non-zero. Thus, $det(\mathbf{M_k})$ is non-zero since $det(\mathbf{M_k}) = det(\mathbf{M_k^{UL}}) det(\mathbf{M_k^{LR}})$ (by Lemma 6).

Equation (3.23) has a unique solution for $\mathbf{a_k}$, because $det(\mathbf{M_k})$ is non-zero (by Lemma 5). If Equation (3.23) has a unique solution for $\mathbf{a_k}$, Equation (3.22) has a unique solution for $\mathbf{a}$, because there is a unique solution for k. But it has been established that there is no unique solutions for $\mathbf{a}$, so assuming that there is a unique solution for k has led to a contradiction. □

Lemma 8 (VSR witness security) *A computationally-bound adversary cannot use the shares of shareholders in any non-authorized subset* $\overline{\mathcal{B}} \notin \Gamma_{\mathcal{P}}^{(m,n)}$ *(*$|\overline{\mathcal{B}}| < m$*) with the witnesses* $W(k)$*,* $W(s_1)$*, …,* $W(s_m)$ *to uniquely determine k.*

PROOF SKETCH: To prove the lemma, I would prove that determining k from the shares of shareholders in $\overline{\mathcal{B}}$ and the witnesses $W(k), W(s_1), \ldots, W(s_m)$ is computationally equivalent to the intractable problem of determining k from $W(k)$

Conjecture 1 (VSR security) *A computationally-bound adversary cannot use the shares of shareholders in any non-authorized subset* $\overline{\mathcal{B}} \notin \Gamma_{\mathcal{P}}^{(m,n)}$ *(*$|\overline{\mathcal{B}}| < m$*), the shares of shareholders in any non-authorized subset* $\overline{\mathcal{B}'} \notin \Gamma_{\mathcal{P}'}^{(m',n')}$ *(*$|\overline{\mathcal{B}'}| < m'$*), and the witnesses* $W(k), W(s_1), \ldots, W(s_m)$ *to uniquely determine k.*

PROOF SKETCH: From Lemma 7, the adversary cannot uniquely determine k from the shares of shareholders in $\overline{\mathcal{B}}$ and $\overline{\mathcal{B}'}$. From Lemma 8, the adversary cannot uniquely determine k from the shares of shareholders in $\overline{\mathcal{B}}$ and $W(k), W(s_1), \ldots, W(s_m)$.

3.3 The mobile adversary and the VSR protocol

Here, I show that the VSR protocol fulfills requirements from Chapter 2 for a recovery protocol.

- It generates new shares for the next epoch such that they can be used to reconstruct the secret (Theorem 1), and such that they cannot be combined with shares from the current epoch to reconstruct the secret (Lemma 7).
- It includes mechanisms to prevent the adversary from corrupting protocol execution, because the adversary may still control some servers (*i.e.*, shareholders) during the update phase. The old servers broadcast witnesses for the SHARES-VALID and SUBSHARES-VALID conditions (step 2 of REDIST in Figure 3.3). The new servers then verify that the SHARES-VALID and

SUBSHARES-VALID conditions hold before they generate new shares (step 3); if the conditions hold, the new shares are valid (Theorem 1). The protocol does not reconstruct the original secret, thus there is no server the adversary can subvert to obtain the secret directly.

- It erases the shares for the current epoch from server memories (step 4 of REDIST), to prevent the adversary from ever obtaining any other current shares it needs.
- It must allow the system to change the threshold parameter of the underlying data distribution scheme. The system can accomplish such a change when executing the protocol (step 1 of REDIST).

I show how an abstract storage system with dynamic membership can use the VSR protocol to counteract a mobile adversary. Consider the system shown in Figure 3.4. The figure shows the memory contents of the servers and the adversary during two consecutive epochs t and $t+1$ and the intervening update phase. The system uses Shamir's threshold sharing scheme [49], with the parameters (m,n) in epoch t and (m',n') in epoch $t+1$. For simplicity, assume that the sets of servers in the two epochs are disjoint. In each epoch, the adversary may only control a sub-threshold number of servers.

Initially, in epoch t, the adversary has subverted servers 1 through $m-1$ and has obtained their shares. During the update phase, m correct servers from epoch t (say, m through $2m-1$) use the VSR protocol to redistribute their shares to the servers from epoch $t+1$; note that the adversary still controls servers 1 through $m-1$, but does not control any of the new servers $1'$ through n'. At the end of the update phase, servers from epoch t erase their shares. In epoch $t+1$, the adversary subverts servers $1'$ through $m'-1$ and obtains their shares.

The VSR protocol, by fulfilling the design requirements for a recovery protocol, is able to prevent the mobile adversary from ever obtaining enough shares to reconstruct the original data. In epoch $t+1$, the adversary will have the shares from all of the servers it subverted in all epochs, *i.e.*, at most $m-1$ shares from epoch t and $m'-1$ shares from epoch $t+1$. However, it is unable to combine the two sets of shares to reconstruct k (Section 3.2.5). The adversary can never obtain more than $m-1$ shares from epoch t because all of the correct servers from epoch t erase their shares (step 4 of REDIST). There is no server that the adversary can subvert to obtain k directly because the protocol does not reconstruct the secret at any server. Finally, the adversary cannot learn k directly from the broadcast of $W(k)$ (step 2 of REDIST), consistent with the assumption that inversion of the witness function is intractable.

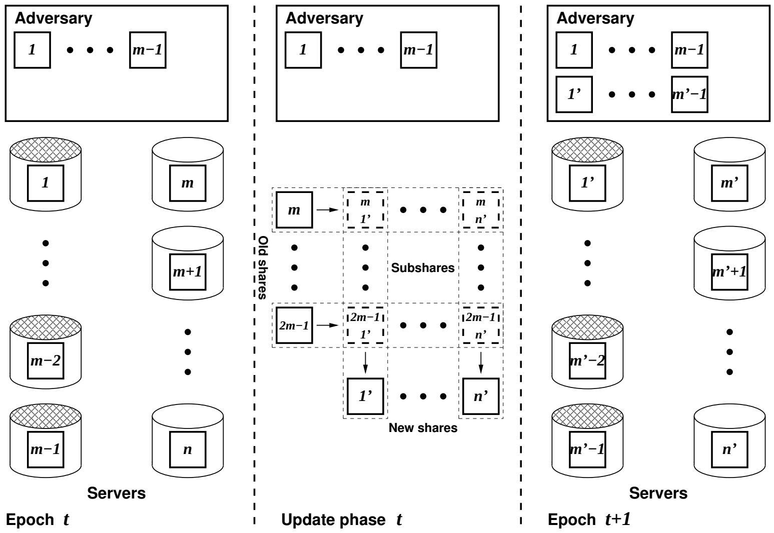

Figure 3.4: A storage system with dynamic membership that uses the VSR protocol to redistribute shares of a secret from the access structure $\Gamma_{\mathcal{P}}^{(m,n)}$ to the access structure $\Gamma_{\mathcal{P}'}^{(m',n')}$, in the presence of a mobile adversary. The memory contents (*i.e.*, shares) of the adversary and servers are shown for two consecutive epochs t and $t+1$ and the intervening update phase. The adversary may only control $m-1$ servers in epoch t, and $m'-1$ servers in epoch $t+1$. Crosshatched servers are under the control of the adversary. The sets of servers in epochs t and $t+1$ are disjoint. The system executes the VSR protocol during the update phase to generate new (marked by primes) shares for correct servers. New shares cannot be combined with current shares to reconstruct the original data. The distribution of subshares during the update phase is shown, but the broadcast of witnesses is omitted to simplify the figure. In epoch t, the adversary subverts servers 1 through $m-1$ and obtains their shares. During the update phase, the system removes all servers from under the control of the adversary. In epoch $t+1$, the adversary subverts servers $1'$ through $m'-1$ and obtains their shares. However, the adversary does not (and can never) obtain enough shares (current or new) to reconstruct the original data.

3.4 Summary

I have presented the VSR protocol for Shamir's sharing scheme. The VSR protocol operates by having an authorized subset of old shareholders distribute subshares of their shares to new shareholders. The new shareholders generate new shares from the subshares. A key insight embodied in the VSR protocol is that the verification is a two-step process: not only must the new shareholders verify the validity of the subshares they receive (the SUBSHARES-VALID condition), but they must also verify the validity of the shares used to distribute those subshares (the SHARES-VALID condition). To enable verification of the validity of old shares, the dealer of the original secret must provide a witness of the secret to the old shareholders during the distribution of shares, and the old shareholders in the authorized subset must all broadcast the witness to the new shareholders. The old shareholders must also broadcast witnesses of their shares, and of the coefficients of the subshare generation polynomials. By verifying the validity of shares and subshares, new shareholders implicitly verify the validity of their new shares (the NEW-SHARES-VALID condition).

I have shown how a storage system can use the VSR protocol to counteract the mobile adversary in a system with dynamic membership. If a faulty old shareholder (*i.e.*, a shareholder under the control of the adversary) sends invalid protocol information, the new shareholders will detect that some subset of the old shareholders are faulty. The new shareholders in some cases can pinpoint faulty shareholders that send invalid information (*e.g.*, if a faulty shareholder sends an invalid subshare), but otherwise can only detect that faulty shareholders exist.

Chapter 4

Hathor: An Experimental Storage System

O thou beautiful Being, thou dost renew thyself in thy season in the form of the Disk, within thy mother Hathor.

— Papyrus of Nekht, Brit. Mus. No. 10471, Sheet 2

In this chapter, I discuss the design and implementation of an experimental storage system called *Hathor*[1]. The primary purpose of Hathor is to prove the thesis that recovery can be efficient. It is an experimental platform for the evaluation of the end-to-end cost of storing, redistributing, and retrieving data, using a variety of data distribution schemes. Hathor is designed to operate in the system environment defined by the abstract system model described in Chapter 2: a client-server environment in which clients are always correct, but in which some number of servers may be controlled by an adversary—that is, they may be faulty. In what follows, I use "the client" as a shorthand to refer to the portion of Hathor that executes on clients in the system, and "the server" as a shorthand to refer to the portion that executes on servers.

[1]Hathor is the Greek transliteration of the Egyptian name *Ht-Hrt*. In Egyptian mythology, Hathor is the Goddess of the Dead and the Protectoress of the City of the Dead in Thebes. I wanted to pick a system name that (for once) was not a Greek or Roman name. Also, Hathor was originally conceived as a write-infrequently, read-mostly data archive prototype, which made the name seem particularly appropriate.

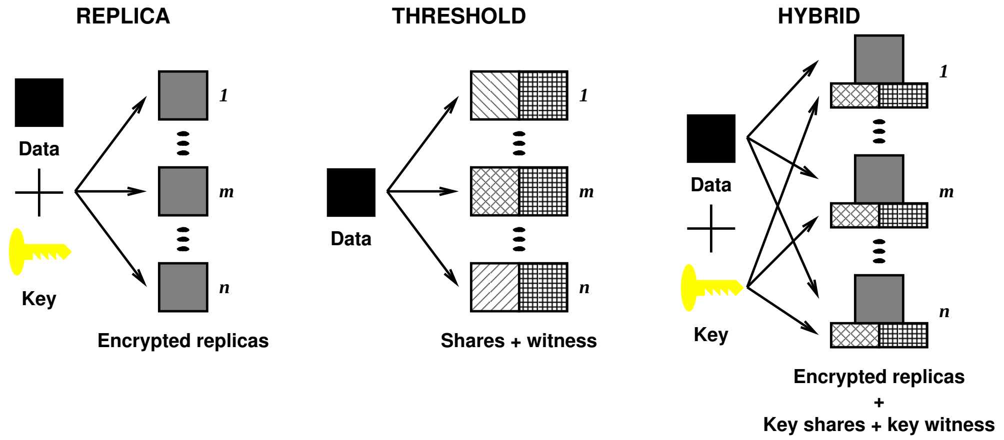

Figure 4.1: REPLICA, THRESHOLD, and HYBRID data distribution schemes implemented in Hathor. In each scheme, n servers stores a piece of the original data, and m pieces are required to recover the data. For both REPLICA and HYBRID, the servers store identical encrypted replicas of the data; additionally for HYBRID, the servers store different shares of the encryption key but identical key witnesses. For THRESHOLD, the servers store different shares of the data but identical witnesses to the data. Each replica, data share, and data witness is the same size as the original data; each key share and key witness is the same size as the key.

4.1 Data distribution schemes

Hathor implements three data distribution schemes: REPLICA, THRESHOLD, and HYBRID. I have selected these schemes in order to model common schemes used in other survivable storage systems. Thus, the performance evaluation in Chapter 5 should provide a rough guide to the cost of supporting decentralized recovery in a variety of environments.

All of the schemes are m-of-n schemes. Figure 4.1 shows the contents of the pieces stored by each server given a particular scheme. To store data, a client distributes a *piece* of the data to each of the n servers; each piece contains a replica, a share, and a witness as appropriate for the distribution scheme in use. To retrieve the data, the client retrieves the pieces from m servers and reconstructs the data. In terms of the mobile adversary model, I assume that storage and retrieval operations occur during an epoch (and not during a update phase), and that an adversary can subvert at most $m-1$ servers in an epoch.

Informally, the process of distribution and reconstruction of data for each scheme is as follows:

- REPLICA: To store data, the client distributes an encrypted replica of the data to each server, and stores the encryption key locally. Each server stores the encrypted replica. To retrieve data, the client retrieves replicas from m servers, verifies that all of the replicas are identical (to prevent faulty servers from convincing it to accept an invalid replica), and decrypts one of the replicas to recover the original data.

REPLICA is similar to the scheme used in the Farsite [1, 11], PAST [47], and Pond [37, 46] survivable storage systems.

- THRESHOLD: To store data, the client uses the INITIAL phase of the VSR protocol (Figure 3.3) to generate n shares of the data and a witness of the data. Each server stores a share and the witness; the servers will use the witness during the redistribution of shares (the REDIST phase of the VSR protocol). To retrieve data, the client retrieves shares from m servers, and uses Shamir's scheme [49] to reconstruct the original data.

  A variant of THRESHOLD without redistribution is implemented as one of the available schemes in the PASIS survivable storage system [53].

- HYBRID: To store data, the client distributes an encrypted replica of the data to each server. It also uses the INITIAL phase of the VSR protocol to generate n key shares and a key witness. Each server stores the encrypted replica, a key share, and the key witness; the servers will use the witness during the redistribution of key shares. To retrieve data, the client retrieves shares and replicas from m servers, verifies that all of the replicas are identical (to prevent faulty servers from convincing it to accept an invalid replica), and uses Shamir's scheme to reconstruct the original encryption key. It then decrypts one of the replicas to recover the original data.

  A variant of HYBRID without redistribution of key shares is similar to the scheme used in the Publius robust publishing system [52]. It is also similar to the scheme used in the e-Vault data repository [33]; the difference is that e-Vault stores IDA fragments [42] of the original data at each server, instead of a complete replica.

Previous comparison studies show that REPLICA is generally faster than either THRESHOLD or HYBRID [53], which raises the question as to why one would use schemes other than REPLICA. One reason is that diversity in schemes can provide an additional defense against system compromise, in the same way that diversity in server operating systems provides a defense: an adversary may not be able to leverage the ability to compromise data stored with one scheme (e.g, REPLICA) to compromise data stored with a different scheme (e.g., THRESHOLD). A second reason is that THRESHOLD and HYBRID enable the client to store data without the need manage an encryption key: with THRESHOLD, no key is required, while with HYBRID, the key is managed with the data.

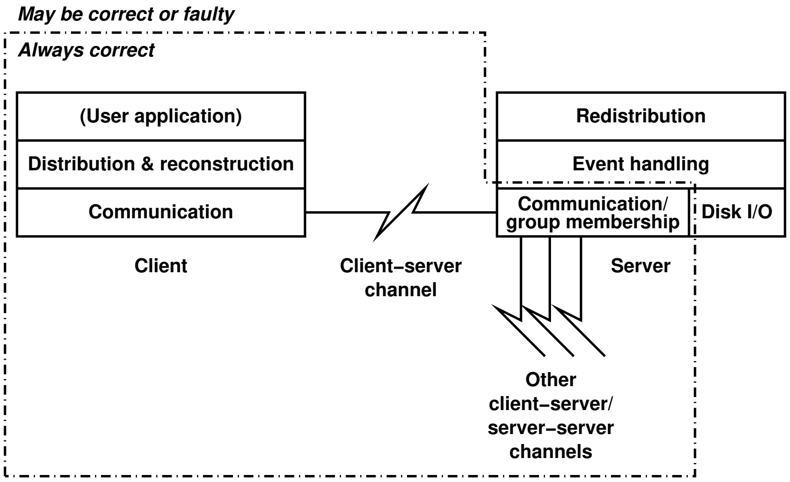

Figure 4.2: Functional modules of the client and server implementations of Hathor. The broken line represents the border between parts of Hathor that are always correct, and parts of Hathor that may be either correct or faulty (i.e., the parts that may be controlled by an adversary). The user application is shown as a module (though it is not strictly part of Hathor), and is included inside the border to emphasize the fact that the application can only access the servers via the distribution and reconstruction module. The various point-to-point channels are also included inside the border.

Though I will use REPLICA as the baseline scheme for the performance evaluation in Chapter 5, one should bear in mind the non-performance reasons for using THRESHOLD or HYBRID.

4.2 Client and server implementation

The Hathor client and server implementations are fairly lightweight. The implementation supports the REPLICA, THRESHOLD, and HYBRID schemes as described above, and also supports the protocols necessary to redistribute pieces of data generated by the schemes. It supports enough of a communications and disk storage infrastructure to measure the end-to-end cost of data storage, redistribution, and retrieval operations.

The client and server implementations in Hathor are composed of abstract functional modules, as shown in Figure 4.2. The dashed line around the modules represents the border between the parts of the Hathor that are always correct and the parts that may potentially be faulty (i.e., the modules that may be controlled by an adversary). I include the various point-to-point channels inside the border to emphasize the assumptions about their privacy, reliability, and ordering properties.

The Hathor client implements the functionality necessary to store and retrieve data. The distribution and reconstruction module implements the threshold sharing and encryption algorithms used by REPLICA, THRESHOLD, and HYBRID, and the communication module manages the client-server channels. A user application stores and retrieves data through an interface that is exported by the distribution module. I assume that the application can only access the servers via the distribution module, and thus cannot disrupt server operation through the direct injection of invalid messages into the client-server channel. To emphasize this constraint, I show the application as a module in the client in Figure 4.2, and include it inside the border of the parts of Hathor that are always correct.

The Hathor server is more complex than the client. The redistribution module implements a pair of state machines (Section 4.3.2) to manage the redistribution of pieces of data for REPLICA, THRESHOLD, and HYBRID. A disk I/O module marshals pieces of data to and from the on-disk representations. An event module parses the messages that drive the redistribution state machine; it also passes client requests to store and retrieve pieces to the disk I/O module.

The server communication module is largely a wrapper around the Ensemble group communications toolkit [29]. The module manages the client-server channels, as well as the server-server channels. The module (in conjunction with the communication modules on other servers) also implements a broadcast channel abstraction on top of the server-server channels. A group membership service within the module maintains a *view* of active servers that are sending and receiving broadcast messages. The service updates the view whenever a server joins or leaves Hathor, and assigns a unique *rank* to each server in the view. The communication module is included inside the border of modules that are always correct, consistent with the assumption in Chapter 2 that an adversary cannot disrupt the broadcast channel (e.g., a set of servers controlled by the adversary cannot break the property that a broadcast message M is either delivered to all servers or to none of them). Note that the adversary can still see all broadcast messages even if the communication modules are always correct: a server controlled by the adversary can simply pass on any messages received by the server at the event module.

4.3 I/O operations

Hathor supports three basic I/O operations: STORE, REDISTRIBUTE, and RETRIEVE. STORE stores pieces of data to a set of servers. RETRIEVE retrieves a sufficient number of pieces to reconstruct the original data. REDISTRIBUTE redistributes pieces from an old set of servers to the new set of servers, and verifies that any new pieces can be used to reconstruct the original data. In this section, I

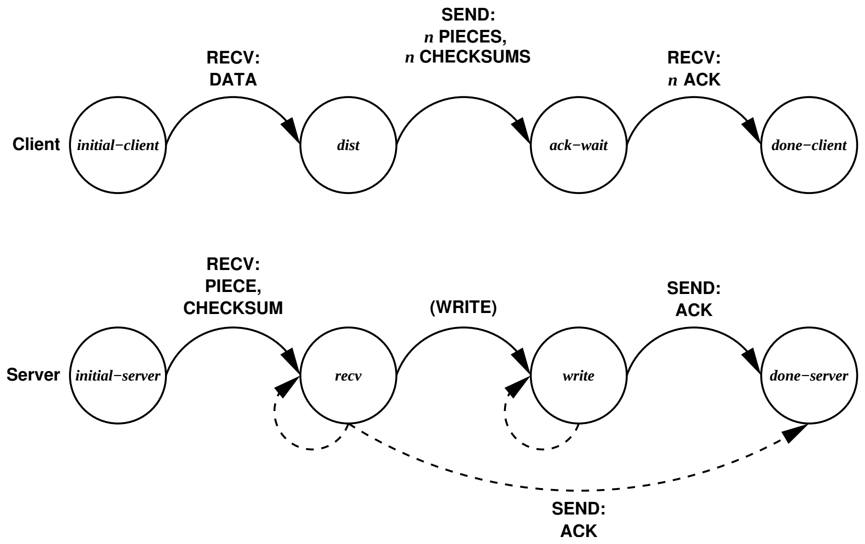

Figure 4.3: STORE state machine for clients and servers. States correspond to intermediate computations, while state transitions correspond to sent or received messages. Clients start in the *initial-client* state, and servers start in the *initial-server* state. Solid arrows indicate transitions taken by correct clients and servers, while dashed arrows indicate transitions that may be taken by faulty servers. Unlabeled dashed arrows that loop to the same state indicate no-op self-transitions.

discuss what steps each operation takes for each data distribution scheme. In particular, I show how the implementation of REDISTRIBUTE for the THRESHOLD scheme corresponds to the VSR protocol in Section 3.2, and how the implementations of REDISTRIBUTE differ between the REPLICA, THRESHOLD, and HYBRID schemes (which will help to explain the results in Section 5.3).

For each scheme, I consider storage and retrieval using an m-of-n scheme, and redistribution to change the scheme parameters from m-of-n to m'-of-n'. As stated before, I assume during a STORE or RETRIEVE that at most $m-1$ servers may be faulty. Additionally, I assume during a REDISTRIBUTE that at least m old servers are correct, that at most $m-1$ old servers are faulty, that at least m' new servers are correct, and that at most $m'-1$ new servers are faulty, consistent with the assumptions about faulty servers in the presentation of the VSR protocol.

4.3.1 STORE

The state machines in Figure 4.3 show the sequences of steps at the client and server for a STORE to an m-of-n scheme. Transitions between states are atomic, *e.g.*, the client does not make a transition from the *ack-wait* state to the *done-client* state until it has received all n acknowledgments. I describe the sequence of steps for a client, a correct server, and a faulty server below.

A client starts in the *initial-client* state. When the client receives data from the user application, it makes a transition to the *dist* state, in which it obtains the view of active servers from the servers' group membership service, and distributes a piece of the data to each of the servers. The client also computes and sends an checksum of the data to each server; it will later use the checksum to verify the validity of reconstructed data. The client next makes a transition to the *ack-wait* state, in which it waits to receive an acknowledgment from each of the n servers; an acknowledgment indicates that a server has written its piece to disk. The client then makes a final transition to the *done-client* state.

A correct server starts in the *initial-server* state. When the server receives a piece of data and a checksum from a client, it makes a transition to the *recv* state. In the *recv* state, the server "sends" a write request to itself, and immediately makes a transition to the *write* state. In the *write* state, the server writes the piece and the checksum to disk, sends an acknowledgment back to the client, and makes a transition to the *done-server* state.

A faulty server also starts in the *initial-server* state. When the server receives a piece of data from a client, it makes a transition to the *recv* state. In the *recv* state, a faulty server may send an acknowledgment to the client, and make a transition to the *done-server* state. The client will think the server has stored its piece even though the server has not written the piece to disk. However, m servers are guaranteed to write their pieces to disk (consistent with the assumption that at least m servers are correct), thus subsequent REDISTRIBUTE or RETRIEVE operations can execute.

A faulty server may also cease to send messages while in the *recv* or *write* states, which results in no-op self-transitions. If the server remains in these states forever, the client that sent the piece (that triggered the transition from the *initial-server* state to *dist* state) will wait forever to receive an acknowledgment from the server, and will thus wait forever to complete a STORE.

Strictly speaking, a client does not need to wait to receive acknowledgments from the servers, because the servers are guaranteed to receive pieces sent by the client (consistent with the assumption of reliable private channels in Chapter 2), and at least m correct servers are guaranteed to write their pieces to disk. In practice, the client waits to receive acknowledgments to measure of the end-to-end cost of storing data: the client starts its timer when it receives data from the application, and stops its timer when it receives acknowledgments from the n servers. The client thus runs the risk that it will wait forever in the *ack-wait* state if a faulty server does not send an acknowledgment.

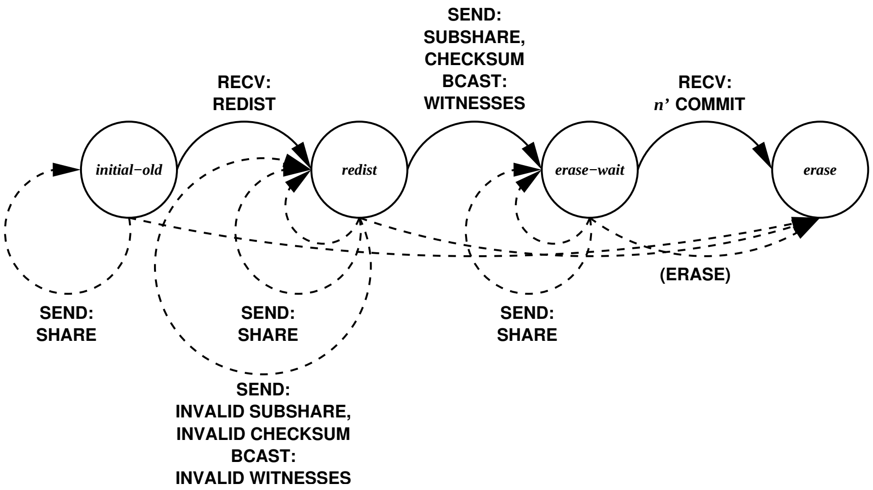

Figure 4.4: REDISTRIBUTE state machine for old servers. States correspond to intermediate computations, while state transitions correspond to sent or received messages. Old servers start in the *initial-old* state. Solid arrows indicate transitions taken by correct servers, while dashed arrows indicate transitions that may be taken by faulty servers. Unlabeled dashed arrows that loop to the same state indicate no-op self-transitions.

4.3.2 REDISTRIBUTE

The state machines in Figure 4.4 and Figure 4.5 show the sequences of steps for a REDISTRIBUTE when changing scheme parameters from m-of-n to m'-of-n', in which m old servers redistribute their pieces of the original data to n' new servers. There is no client state machine because clients do not participate in REDISTRIBUTE. As for STORE, transitions between states are atomic, *e.g.*, an old server does not make the transition from the *erase-wait* state to the *erase* state until it has received all n' "commit" messages. For simplicity, I describe the steps when THRESHOLD is used to distribute pieces of the original data, and then highlight how REPLICA or HYBRID differ.

In the current implementation of Hathor, m old servers and all n' new servers participate in REDISTRIBUTE. Moreover, Hathor requires all n' new servers to send "commit" messages before completing redistribution, to ensure that all non-faulty servers have valid pieces of data; thus, a faulty old server or a faulty new server can cause all servers to either hang or abort. In an experimental system whose primary purpose is to measure the end-to-end cost of redistribution, such behavior is tolerable: a human operator can shut Hathor down and determine the cause of the fault. However, such behavior would be unacceptable for a production system.

Old server state machine

A correct old server follows the state machine transitions indicated by the solid arrows in Figure 4.4. The server starts in the *initial-old* state with a share of the original data and a witness of the data. When the server receives a redistribution request, it makes a transition to the *redist* state. There, the server distributes subshares of its share (step 1 of REDIST in the VSR protocol of Figure 3.3) and the checksum of the original data to each server. The server also broadcasts the witness of the data, the witness of its share, and the witnesses of the coefficients of its subshare generation polynomial (step 2 of REDIST). Once the server has sent subshares and witnesses, it makes a transition to the *erase-wait* state. In the *erase-wait* state, the server waits to receive a "commit" message from each of the n' new servers. Upon receiving the n' "commit" messages, the server makes a final transition to the *erase* state, and erases its share.

A faulty old server may make the following additional state machine transitions, as indicated by the dashed arrows in Figure 4.4:

- It may send its share to the adversary while in any state other than the *erase* state. Sending a share results in a self-transition. The *erase* state does not have this self-transition since the server will have erased its share in this state.

  Note that the adversary will never obtain enough shares to reconstruct the original data, consistent with the assumption that at most $m-1$ old servers are faulty.

- It may "send" an erase request to itself while in any state other than the *erase* state, which results in a transition to the *erase* state.

- It may send invalid subshares, witnesses, or checksums while in the *redist* state. Recall from Section 3.2.1 that invalid subshares do not reconstruct a consistent value, and that invalid witnesses do not correspond to the original data, or to shares of the data, or to the coefficients of the subshare generation polynomial. Sending invalid information results in a self-transition: the server remains in the *redist* state.

- It may refuse to send subshares or witnesses while in the *redist* state, and thus remain in that state forever (resulting in a no-op self-transition).

- It may refuse to "send" an erase request to itself while in the *erase-wait* state, and thus remain in that state forever (resulting in a no-op self-transition).

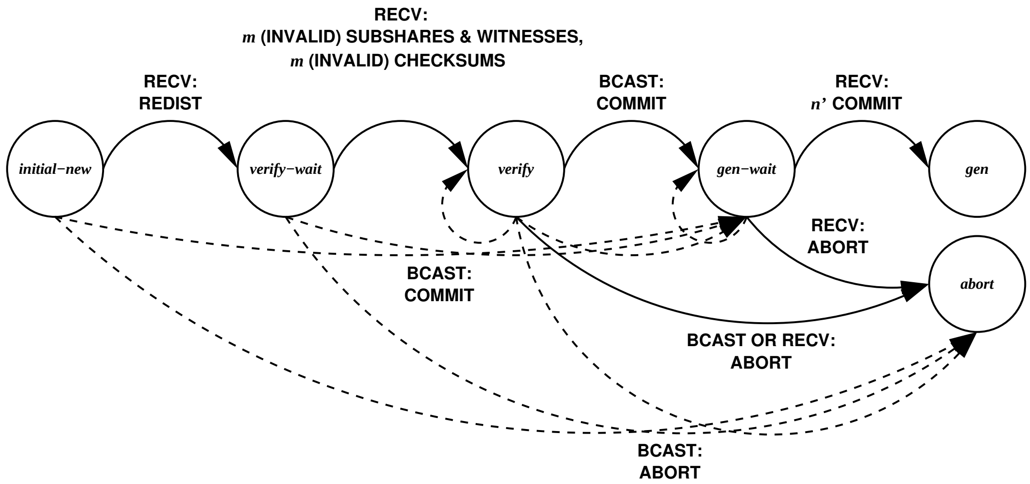

Figure 4.5: REDISTRIBUTE state machine for new servers. States correspond to intermediate computations, while state transitions correspond to sent or received messages. New servers start in the *initial-new* state. Solid arrows indicate transitions taken by correct servers, while dashed arrows indicate transitions that may be taken by faulty servers. Unlabeled dashed arrows that loop to the same state indicate no-op self-transitions.

New server state machine

A new server follows the state machine transitions indicated by the solid arrows in Figure 4.5. The server starts in the *initial-new* state. When the server receives a redistribution request, it makes a transition to the *verify-wait* state, in which the server waits to receive a subshare, the corresponding witnesses, and a checksum from each of the m old servers; note that a faulty old server may send invalid information. Upon receiving the m subshares, witnesses, and checksums, the server makes a transition to the *verify* state.

While in the *verify* state, a new server tries to verify that the SHARES-VALID and SUBSHARES--VALID conditions hold for the subshares and witnesses (step 3 of REDIST in the VSR protocol of Figure 3.3). It also tries to verify that all m of the received checksums match (to prevent $m-1$ faulty servers from convincing it to accept an invalid checksum). The server may then make one of several possible transitions:

- If it verifies that the SHARES-VALID and SUBSHARES-VALID conditions hold, and that the checksums match, it broadcasts a “commit” message, which results in a transition to the *gen-wait* state.
- If it finds that the SHARES-VALID and SUBSHARES-VALID conditions do not both hold, or that the checksums do not match, it broadcasts an “abort” message to indicate that it has

received invalid information from at least one of the old servers (i.e., at least one of the m old servers is faulty; recall from Section 3.2.2 that new servers may not be able to pinpoint faulty old servers). Sending an "abort" message results in a transition to the *abort* state.

- If it receives an "abort" message, it makes a transition to the *abort* state.

When the server enters the *gen-wait* state, it waits to receive a "commit" message from each of the n' new servers (note that it will have received its own message). If the server receives an "abort" message, it makes a transition to the *abort* state. Otherwise, upon receiving n' "commit" messages, the server makes a transition to the *gen* state, in which it generates a new share from the received subshares, and writes the share, the witness to the original data, and the checksum to disk.

A faulty old server may make the following additional state machine transitions, as indicated by the dashed arrows in Figure 4.5:

- It may broadcast a "commit" message while in any of the *initial-new*, *verify-wait*, or *verify* states, even if it has not verified that the SHARES-VALID and SUBSHARES-VALID conditions both hold. This results in a transition to the *gen-wait* state.
- It may broadcast an "abort" message while in any of the *initial-new*, *verify-wait*, or *verify* states, even if it has not found that the SHARES-VALID and SUBSHARES-VALID conditions do not both hold. This results in a transition to the *abort* state.
- It may refuse to broadcast either a "commit" or an "abort" message while in the *verify* state, and thus remain in that state forever (resulting in a no-op self-transition).
- It may refuse to generate a new share, and thus remain in the *gen-wait* state forever (resulting in a no-op self-transition).

REDISTRIBUTE for REPLICA

The steps for an old server for REPLICA are slightly different from that for THRESHOLD. In the state machine in Figure 4.4, an old server starts in the *initial-old* state with an encrypted replica of the original data. In the *redist* state, the server sends its encrypted replica (instead of subshares and witnesses). A faulty old server may instead send an invalid replica (i.e., one that is not an encryption of the original data). A key difference between REPLICA and THRESHOLD is that no broadcast occurs during *redist*.

The steps for a new server are also slightly different. In the state machine in Figure 4.5, a new server in the *verify-wait* state waits to receive a replica from each of the m old servers. In the *verify* state, the server tries to verify that all m of the received replicas match (to prevent $m-1$ faulty servers from convincing it to accept an invalid replica). If the m replicas match, the server broadcasts a "commit" message, otherwise it broadcasts an "abort" message. In the *gen* state, the server simply writes the replica to disk.

Versions of the old and new server state machines for REPLICA are shown in Figure A.1.

REDISTRIBUTE for HYBRID

The steps for HYBRID are a combination of those for both REPLICA and THRESHOLD. In the state machine in Figure 4.4, an old server starts in the *initial-old* state with an encrypted replica of the original data, a share of the encryption key, and a key witness. In the *redist* state, the server sends its encrypted replica, and also sends subshares of its key share and the corresponding witnesses. A faulty old server may instead send invalid information.

The steps for a new server are also a combination. In the state machine in Figure 4.5, the server in the *verify-wait* state waits to receive a replica, a key subshare, and witnesses from each of the m old servers. In the *verify* state, a server verifies that all m of the received replicas match, and also verifies that the SHARES-VALID and SUBSHARES-VALID conditions hold for the key subshares and witnesses. If the m replicas match and the SHARES-VALID and SUBSHARES-VALID conditions hold, the server broadcasts a "commit" message, otherwise it broadcasts an "abort" message. In the *gen* state, the server generates a new key share from the received key subshares, and writes the share, the key witness, and the replica to disk.

Versions of the old and new server state machines for HYBRID are shown in Figure A.2.

4.3.3 RETRIEVE

The state machines in Figure 4.6 show the sequences of steps at the client and server for a RETRIEVE from an m-of-n scheme. Transitions between states are atomic, *e.g.*, the client does not make a transition from the *reconst-wait* state to the *reconst* state until it has received all m shares, checksums, and replicas (as appropriate for the distribution scheme). I describe the sequence of steps for a client, a correct server, and a faulty server below.

A client starts in the *initial-client* state. When the client receives a retrieval request for the

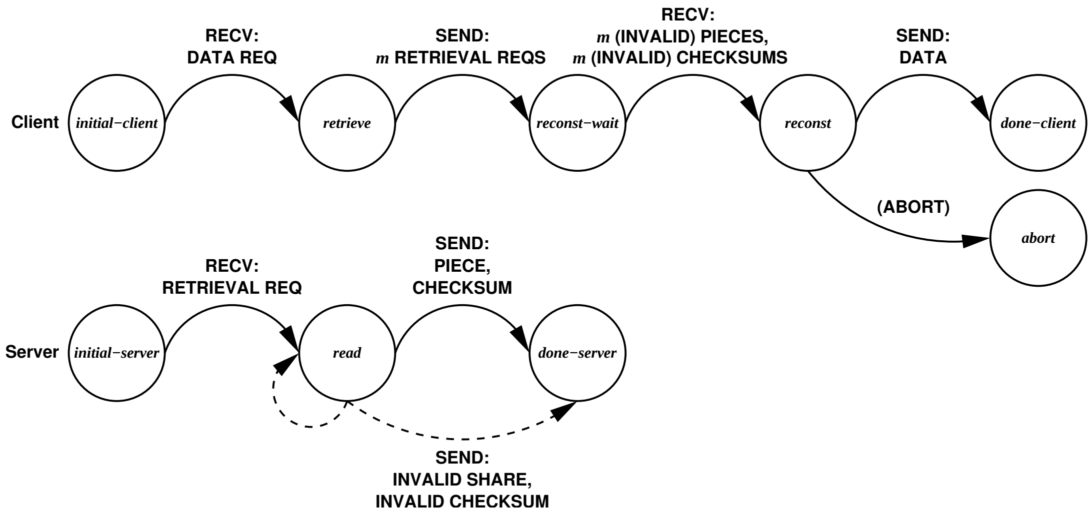

Figure 4.6: RETRIEVE state machine for clients and servers. States correspond to intermediate computations, while state transitions correspond to sent or received messages. Clients start in the *initial-client* state, and servers start in the *initial-server* state. Solid arrows indicate transitions taken by correct clients and servers, while dashed arrows indicate transitions that may be taken by faulty servers. Unlabeled dashed arrows that loop to the same state indicate no-op self-transitions.

original data from a user application, it makes a transition to the *retrieve* state, in which it obtains the view of active servers from the servers' group membership service, and sends a retrieval request to the m highest ranked servers for their pieces of the data and the checksum of the data. The client then makes a transition to the *reconst-wait* state, in which it waits to receive a piece and checksum from each of the m servers. Upon receiving the m pieces and checksums, the client makes a transition to the *reconst* state, in which it reconstructs the data from the pieces as described in Section 4.1. It also verifies that all m of the received checksums match. If the checksums do not match, or the checksum of the reconstructed data does not match the retrieved checksum, the client makes a transition to the *abort* state. Otherwise, it sends the original data back to the user application, and makes a transition to the *done-client* state.

A correct server starts in the *initial-server* state. When the server receives a retrieval request from a client, it makes a transition to the *read* state, in which it reads its piece and checksum of the original data from disk, and sends the piece and checksum to the client. It then makes a transition to the *done-server* state.

A faulty server also starts in the *initial-server* state. When the server receives a retrieval request, it makes a transition to the *read* state. In the *read* state, a faulty server may send an invalid piece or an invalid checksum to the client. It then makes a transition to the *done-server* state.

A faulty server may also cease to send messages while in the *read* state, which results in a no-op self-transition. If the server remains in this state forever, the client that requested the piece of data will wait forever for a piece from the server, and will thus wait forever to complete a RETRIEVE.

In the current implementation of Hathor, RETRIEVE suffers from a similar weakness as REDISTRIBUTE: one or more faulty servers can cause the operation to hang. In a production system, the client could retrieve pieces from at least m servers and try each m size permutation the pieces until it found a permutation that contained only valid pieces.

4.4 Summary

I have presented the design and implementation of Hathor, an experimental storage system for evaluating the cost of storing, redistributing, and retrieving data. The presentation includes details of the REPLICA, THRESHOLD, and HYBRID data distribution schemes, as well as client and server state machine specifications for the STORE, REDISTRIBUTE, and RETRIEVE operations.

Chapter 5

Performance Evaluation

I've got signals! I've got readings—in front and behind!

— Pvt. William Hudson, in "Aliens" (1986)

In this chapter, I present the results of experiments to measure the raw computation performance of the VSR protocol (Chapter 3). The results show that the cost of the VSR protocol is dominated by the cost of the witness function used in verification operations. I also present the results of experiments to measure the end-to-end cost of storing, redistributing, and retrieving data with the Hathor storage system (Chapter 4), using the REPLICA, THRESHOLD, and HYBRID data distribution schemes (Section 4.1). The results show that although the overhead of THRESHOLD over REPLICA increases with file size for all storage operations, the overhead of HYBRID remains roughly constant. I go on to discuss ways of reducing the overhead still further.

In the presentation of the VSR protocol, I treated secrets as integers drawn from a finite field with prime modulus p (Section 3.1.1). In practice, blocks of data cannot be treated as a single integer: to treat a 1024-byte block this way would require a 8193-bit modulus! Thus, I treat blocks of data as sequences of multi-bit *chunks*. If the modulus has size $|\, p\,|$, then each chunk has size $|\, p\,| - 1$ bits, to ensure that the integer representation of a chunk is less than p. We will see shortly that $|\, p\,|$, and hence the chunk size, can have a counterintuitive effect on performance.

All performance numbers were taken on 2 GHz Intel Pentium 4 CPU workstations with 512 MB of RAM. Arbitrary-precision integer operations were implemented using the Miracl package [50].

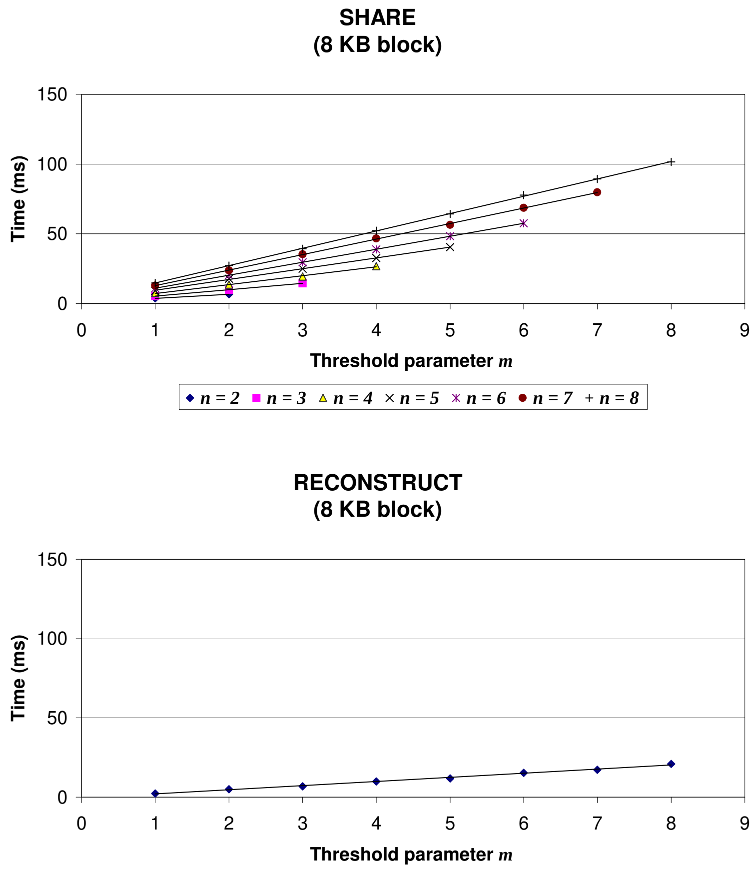

Figure 5.1: Graphs of the time taken to generate n shares of an 8 KB secret, and reconstruct an 8 KB secret from m shares, when using Shamir's (m, n) threshold sharing scheme [49].

5.1 Shamir's threshold sharing scheme performance

In this section, I present the results of experiments to measure the cost of Shamir's (m, n) threshold sharing scheme [49] in a finite field. Sharing operations were computed modulo a 1024-bit prime number p. No witness function operations were performed. The results on their own are unremarkable, but they will serve to show that the bulk of the computation cost of verifiable secret redistribution is in verification-related operations, and not in threshold sharing operations.

The SHARE and RECONSTRUCT graphs in Figure 5.1 show the time taken to generate n shares of an 8 KB secret, and the time taken to reconstruct an 8 KB secret from m shares. For both operations, the time taken is linear in m for fixed n. We observe that the marginal cost for SHARE with increasing n to generate an extra share is relatively small, and is constant for fixed m.

5.2 Verifiable secret redistribution performance

In this section, I present the results of experiments to measure the computation cost of the VSR protocol (Section 3.2). To assist with understanding where the costs are in the VSR protocol, I present the time taken to execute the following steps within the protocol:

- SUBSHARE: The time taken by an old shareholder i to generate subshares $\hat{s}_{ij}$ and SUBSHARES-VALID witnesses $W(s_i)$, $W(a_1)\ldots W(a_{m-1})$ for the new shareholders j (steps 1 and 2 of REDIST in the VSR protocol of Figure 3.3).
- SHARES-VALID: The time taken by a new shareholder j to verify that the SHARES-VALID condition holds (the second verification in step 3).
- SUBSHARES-VALID: The time taken by a new shareholder j to verify that the SUBSHARES-VALID condition holds, which involves computing $W(\hat{s}_{ij})$ (the first verification in step 3).
- GENERATE-NEW-SHARE: The time taken by a new shareholder j to generate a new share s'_j from the subshares (step 4).

The cost of the VSR protocol is dominated by the time taken to compute the witness function $W(x)$ used in SUBSHARE and SUBSHARES-VALID. To investigate the cost, I measured the time taken by each of the protocol steps using the two witness functions introduced in Section 3.1.3: exponentiation and point multiplication in an elliptic curve.

5.2.1 VSR with an exponentiation witness function

Here, I present the performance results for SUBSHARE, SHARES-VALID, SUBSHARES-VALID, and GENERATE-NEW-SHARE with an exponentiation witness function (Section 3.1.3), for redistribution from an (m,n) to an (m',n') threshold scheme. Sharing operations were computed modulo a 1024-bit prime number p. The witness function was exponentiation modulo a 1025-bit prime number q, where $q = pr + 1$ (r being a positive integer); given a q of size $\mid q \mid$ bits, the largest possible p has size $\mid p \mid$ less than or equal to $\mid q \mid -1$. The original secret, shares, and subshares were all 8 KB.

The SUBSHARE and SUBSHARES-VALID steps of the VSR protocol involve repeated exponentiation with a common base g. The Miracl [50] package implements basic square-and-multiply exponentiation, and also implements a method of fast exponentiation (with precomputation of intermediate values) presented by Brickell *et al.* [12]. Brickell exponentiation yields a considerable

$\mid p \mid$	$\mid q \mid$	Basic exp (μs)	Brickell exp (μs)	Brickell precomp (μs)
160	1024	3376	1201	12622
192	1024	3991	1350	15053
1023	1024	19957	5301	62684
257	1025	5445	1627	20619
1024	1025	20283	5486	64408

Table 5.1: Time per exponentiation modulo q for exponents modulo p, for basic square-and-multiply exponentiation and Brickell fast exponentiation [12]. The precomputation time required for Brickell exponentiation is also shown.

performance boost over square-and-multiply (Table 5.1), provided that one amortizes the precomputation over multiple exponentiations. Where possible, I used Brickell exponentiation.

The SUBSHARE graph in Figure 5.2 shows the time taken by an old shareholder i to generate subshares and witnesses from its share for a new (m', n') threshold scheme. The time taken is linear in m' for fixed n', as predicted in the performance model of Section 3.2.3. The generation of subshares (without generation of witnesses) for an (m', n') scheme is mathematically equivalent to generation of shares of a secret for an (m', n') scheme, and therefore has the same performance characteristics as the SHARE operation in Figure 5.1. For fixed m', the marginal cost with increasing n' comes from the time taken to generate an extra subshare.

Given the results for SHARE, and given that generation of subshares is equivalent to generation of shares, it is clear that the bulk of the cost of SUBSHARE comes from the time taken to generate the witness g^{s_i} for the old share s_i. Consider Figure 5.3, which shows two curves for $n' = 8$: the time taken to generate subshares, and the time taken to generate subshares and witnesses. The bulk of the cost of SUBSHARE comes from the time taken to compute the witness g^{s_i} for the old share s_i. This fact is demonstrated by the points at $m' = 1$, at which one only computes g^{s_i} (the corresponding subshare generation polynomial has no coefficients).

The SHARES-VALID graph in Figure 5.2 shows the time taken by a new shareholder j to verify the validity of shares held by the shareholders in an authorized subset of the old (m, n) threshold scheme. The verification was performed using witnesses sent by the first m old shareholders ($i \in \mathcal{B} = \{1, \ldots, m\}$). The time taken has a stepping behavior in m. To explain this, recall the SHARES-VALID verification equation

$$g^k \equiv \prod_{i \in \mathcal{B}} (g^{s_i})^{b_i} \tag{5.1}$$

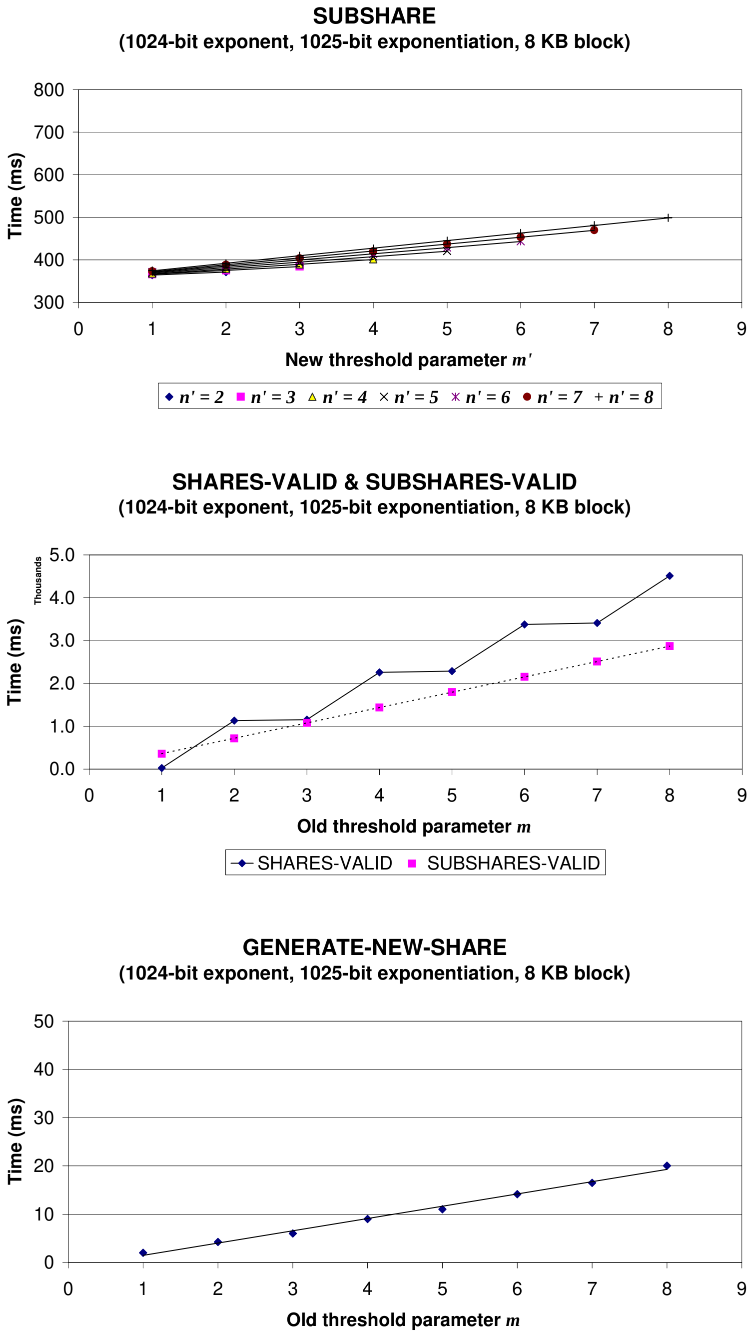

Figure 5.2: Graphs of the time taken by SUBSHARE, SHARES-VALID, SUBSHARES-VALID, and GENERATE--NEW-SHARE with an exponentiation witness function. Redistribution was from an (m,n) threshold sharing scheme to an (m',n') scheme. The original secret, shares, and subshares were all 8 KB. The y-axis scales on the graphs are different because the results are of such different orders of magnitude.

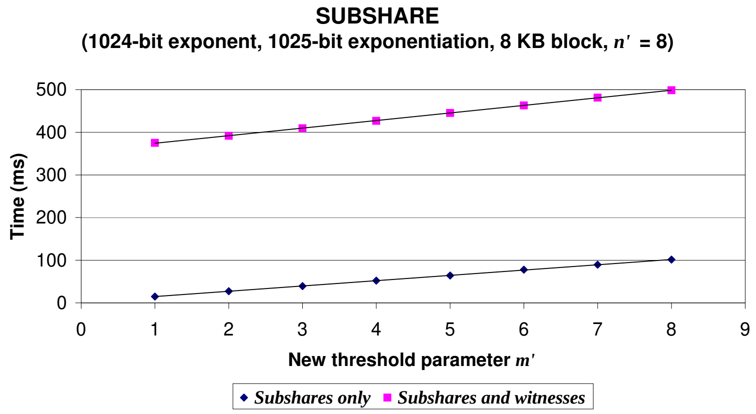

Figure 5.3: Graph of time taken by SUBSHARE with an exponentiation witness function. The graph shows both the time take to generate subshares only, and the time taken to generate subshares and witnesses. Redistribution was from an (m,n) threshold sharing scheme to an (m',n') scheme. The original secret, shares, and subshares were all 8 KB.

(Equation (3.17) for exponentiation), and observe that a new shareholder only exponentiates by the Lagrange coefficients b_i. By experiment, I confirmed that the sum of b_i given m and p is

$$\sum_{i\in\mathcal{B},\mathcal{B}=\{1,\ldots,m\}} b_i = \left\lceil \frac{m-1}{2} \right\rceil p+1 \tag{5.2}$$

i.e., the sum of the exponents in Equation (5.1) is the same for pairs of consecutive m (2 and 3, 4 and 5, etc.), and increases linearly with even values of m. Thus, the cost of computation for Equation (5.1) will be similar for pairs of consecutive m.

The SUBSHARES-VALID graph in Figure 5.2 shows the time taken by new shareholder j to verify the validity of the subshares sent by the shareholders in an authorized subset of the old (m,n) threshold scheme. The time taken is linear in m, as predicted in the performance model of Section 3.2.3. Moreover, the time is smaller that the SHARES-VALID time for all m; to understand this, recall the SUBSHARES-VALID verification equation

$$\forall i \in \mathcal{B} : g^{\hat{s}_{ij}} \equiv g^{s_i} \prod_{l=1}^{m'-1} (g^{a'_{il}})^{j^l} \tag{5.3}$$

and observe that a new shareholder only computes exponentiations of base g by $\hat{s}_{ij}$ (excluding the j^l term); thus, one can use Brickell exponentiation.

$\mid p \mid$	$\mid q \mid$	Exponentiation (μs)	Chunk size (bits)	Chunks/block	Expected time/block (ms)
160	1024	1204	159	413	496
192	1024	1350	191	344	463
1023	1024	5301	1022	65	340
1024	1025	5486	1023	65	351

Table 5.2: Time per exponentiation modulo q for exponents modulo p, using Brickell exponentiation [12]. A block was 8 KB.

The GENERATE-NEW-SHARE graph in Figure 5.2 is unremarkable: because this step is mathematically equivalent to secret reconstruction in Shamir's threshold sharing scheme [49], the results are the same as in Figure 5.1.

Protocol cost versus size of secret

One would expect the cost of the VSR protocol to be linear in the block size, and I ran experiments to confirm that this is the case. The graphs in Figure 5.4 show the time taken by SUBSHARE, SHARES-VALID, SUBSHARES-VALID, and GENERATE-NEW-SHARE as the block size increases, for redistribution from a (4,8) threshold sharing scheme to a (4,8) scheme. The results indeed confirm that the cost is linear in the block size.

Protocol cost versus size of modulus p

At first glance, it would seem that using a prime modulus p of 1024 bits would needlessly incur a high cost during witness generation, or during verification that the SHARES-VALID and SUBSHARES-VALID conditions hold. By comparison, the Digital Signature Standard [51] only specifies p of 160 bits. Consider Table 5.2, which shows the time taken to exponentiate modulo q given exponents modulo p with bit size $\mid p \mid$; note the change in the size of q for 1024-bit p. On the one hand, smaller $\mid p \mid$ indeed lowers the time per exponentiation. On the other hand, one must trade off a lower time per exponentiation against the number of exponentiations required: with a smaller $\mid p \mid$, one can only process a smaller chunk of data at a time. For example, an 8 KB block of data is 64K bits, which is equivalent to either 413 159-bit chunks or 65 1022-bit chunks—an almost sevenfold difference. We see, then, that the expected time to generate a witness for an 8 KB block is greater with the smaller $\mid p \mid$.

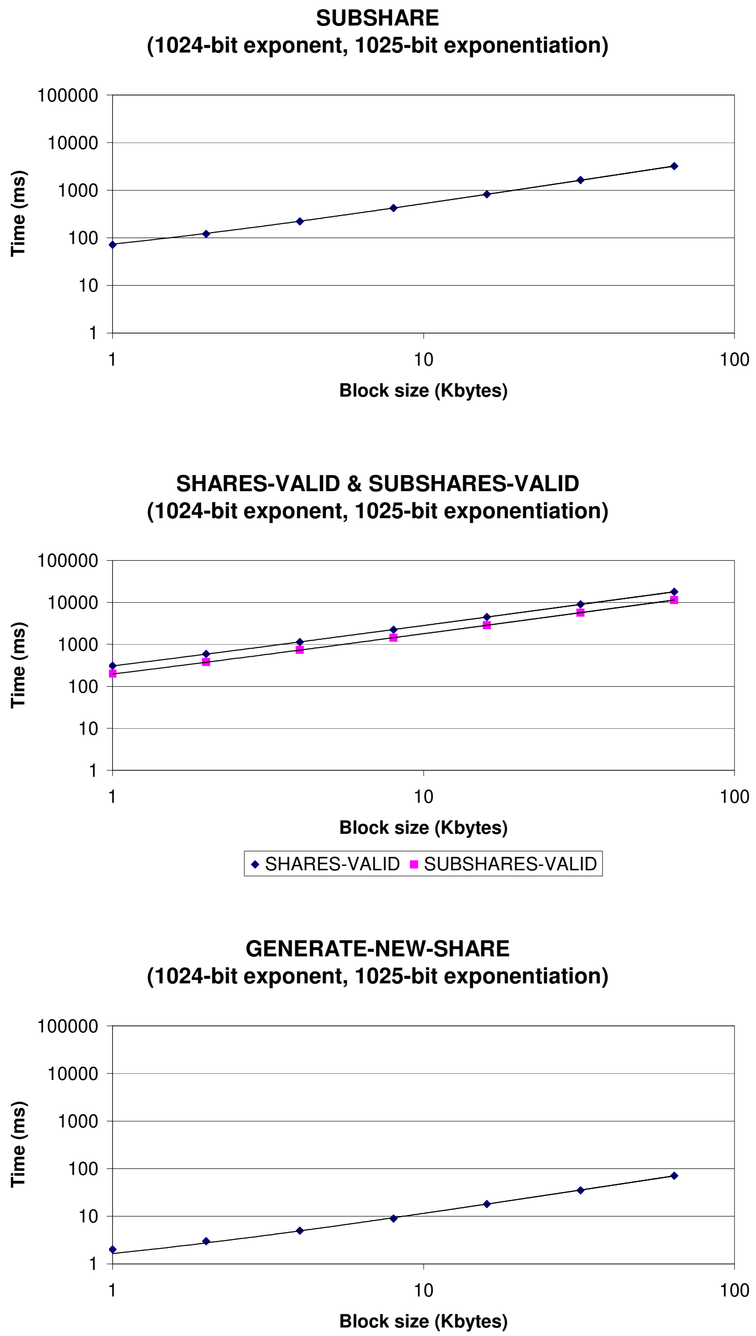

Figure 5.4: Graphs of the time taken by SUBSHARE, SHARES-VALID, SUBSHARES-VALID, and GENERATE--NEW-SHARE with an exponentiation witness function. The time taken is shown as a function of block size. Redistribution is from a (4,8) threshold sharing scheme to a (4,8) scheme.

5.2.2 VSR with an elliptic curve witness function

In this section, I present the performance results by SUBSHARE, SHARES-VALID, SUBSHARES--VALID, and GENERATE-NEW-SHARE with an elliptic curve witness function (Section 3.1.3), for redistribution from an (m,n) to an (m',n') threshold scheme. Sharing operations were computed modulo a 192-bit prime number r. The witness function was multiplication of a public point G on an elliptic curve. The curve was computed over the integers modulo a 192-bit prime number q' (so designated to distinguish it from the modulus for the exponentiation witness function). q' and r were such that $r \leq q'$, and G was such that $[r]G$ was equal to the point at infinity O. The parameters for the underlying elliptic curve and coordinates of G were those specified in the Digital Signature Standard [51] for a 192-bit elliptic curve. The original secret, shares, and subshares were all 8 KB.

The results are similar to those for exponentiation (Section 5.2.2). In particular:

- The SHARES-VALID graph in Figure 5.5 exhibits the same stepping behavior as Figure 5.2 for SHARES-VALID with a exponentiation witness function, for the same reason.
- Brickell exponentiation [12] can be adapted for point multiplication on an elliptic curve [50]. Thus, as with exponentiation, the time taken by SUBSHARES-VALID is lower than than the time taken for SHARES-VALID for all m.

Comparison with exponentiation witness functions

The use of elliptic curves in cryptography was first proposed by Koblitz [35] and Miller [38]. One feature of elliptic curves that makes their use in cryptography attractive is that there are no known sub-exponential algorithms for finding discrete logs on general elliptic curves (though certain curves have exploitable weaknesses; Blake *et al.* [6] present a full discussion of weak curves and attacks). Another is that elliptic curve-based systems require far fewer bits (in the underlying finite field) than exponentiation-based systems to achieve equivalent "security" in terms of the computational effort required to solve the DLP. With fewer bits, one would expect an overall reduction in the cost of a protocol based on the DLP. However, the reduction does not materialize for the VSR protocol.

The reason that the VSR protocol does not benefit from a switch to elliptic curve witness functions is that one must process more chunks for a block of fixed size, similar to the problem discussed in Section 5.2.1. For elliptic curve point multiplication, the chunk size is limited by $\mid r \mid$, which in turn is limited by $\mid q' \mid$; for a 192-bit finite field, the chunk size is 191 bits. Consider Table 5.2 and Table 5.3, and compare multiplication on a curve computed over a 192-bit field to exponentiation

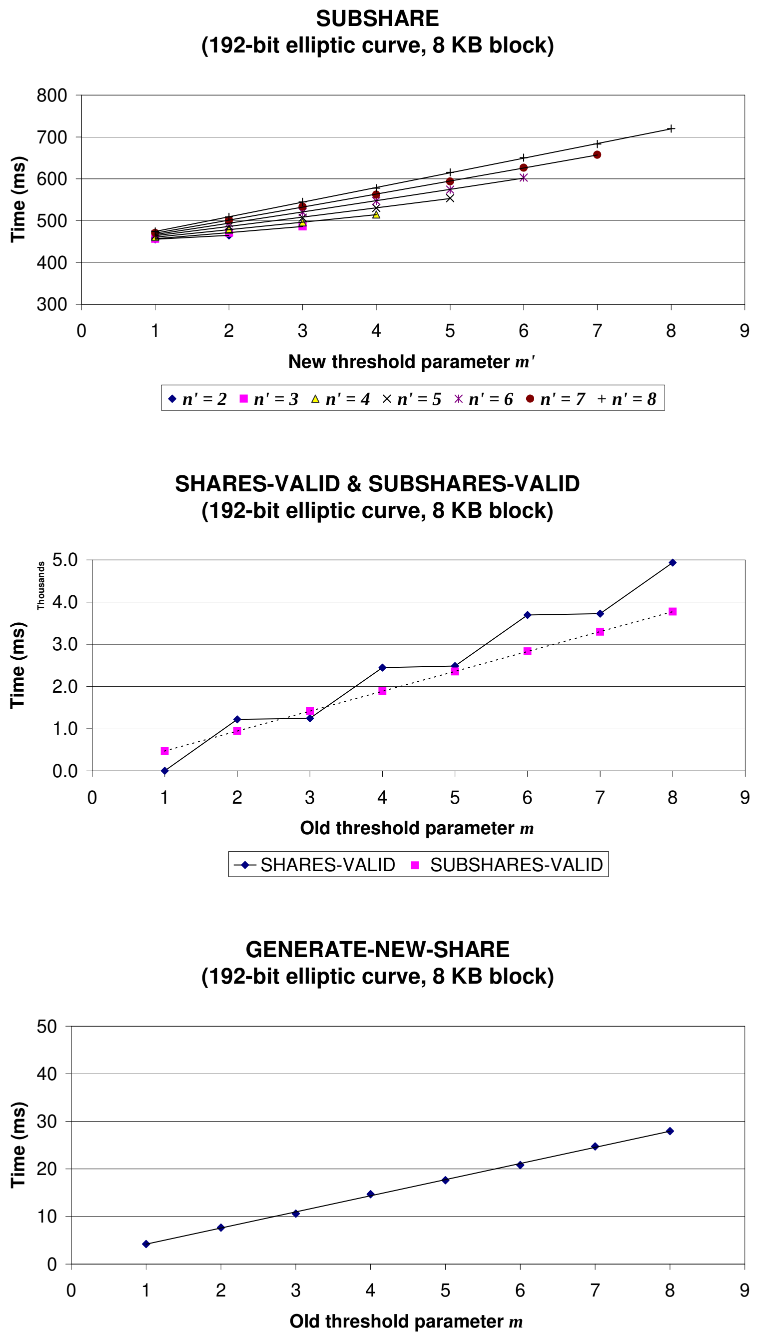

Figure 5.5: Graphs of the time taken by SUBSHARE, SHARES-VALID, SUBSHARES-VALID, and GENERATE-NEW-SHARE with an elliptic curve witness function. The elliptic curve was computed over the 192-bit finite field specified in the Digital Signature Standard [51]. Redistribution was from an (m,n) threshold sharing scheme to an (m',n') scheme. The y-axis scales on the graphs are different because the results are of such different orders of magnitude.

$\mid q' \mid$	Multiplication (μs)	Chunk size (bits)	Chunks/block	Expected time/block (ms)
192	1192	191	344	407
224	1411	223	294	413
256	1742	255	258	446
384	4741	383	172	809
521	8247	520	127	1037

Table 5.3: Time per point multiplication in the elliptic curve computed over the finite field $\mathbb{Z}_{q'}$, using Brickell exponentiation [12]). A block was 8 KB. The elliptic curve parameters were taken from the Digital Signature Standard [51].

in a 1024-bit field (which each require roughly equivalent computational effort to solve the DLP). If we compare multiplication to exponentiation with 1023-bit exponents (*i.e.*, $\mid p \mid = 1023$, the maximum possible size), exponentiation is faster than multiplication: 340 ms/block *vs.* 407 ms/block. The reason for the faster performance of exponentiation is that even though the per-exponentiation cost is higher than the per-multiplication cost by 4.4 times, the number of chunks per block for exponentiation is lower by 5.3 times.

Even though the results suggest that exponentiation yields better performance than point multiplication now, the performance advantage will fall away as the bit size of the underlying fields increases in the future. Figure 5.6 compares the time taken by a new shareholder to verify that the SHARES-VALID condition holds with both exponentiation and elliptic curve witness functions. We see that verification based on exponentiation becomes an order of magnitude slower than the equivalent based on elliptic curves. Thus, for systems that will use larger bit sizes, or to ensure that a system remains secure for the long term, one should use elliptic curve witness functions.

5.3 Hathor storage system performance

In this section, I present the results of experiments to evaluate the Hathor storage system described in Chapter 4. The experiments measured the cost of STORE, REDISTRIBUTE, and RETRIEVE operations for the REPLICA, THRESHOLD, and HYBRID data distribution schemes implemented in Hathor. The results show that the overhead of redistribution incurred by the VSR protocol can be small, provide one is careful in choosing the data distribution scheme.

The experimental infrastructure comprised seven workstations that ran the Hathor server daemon, and one workstation that ran an experiment client driver. The client and servers were connected by a switched 100baseT Ethernet network, which provided full link bandwidth between all pairs of workstations. The client driver initiated STORE and RETRIEVE operations, and also triggered RE-

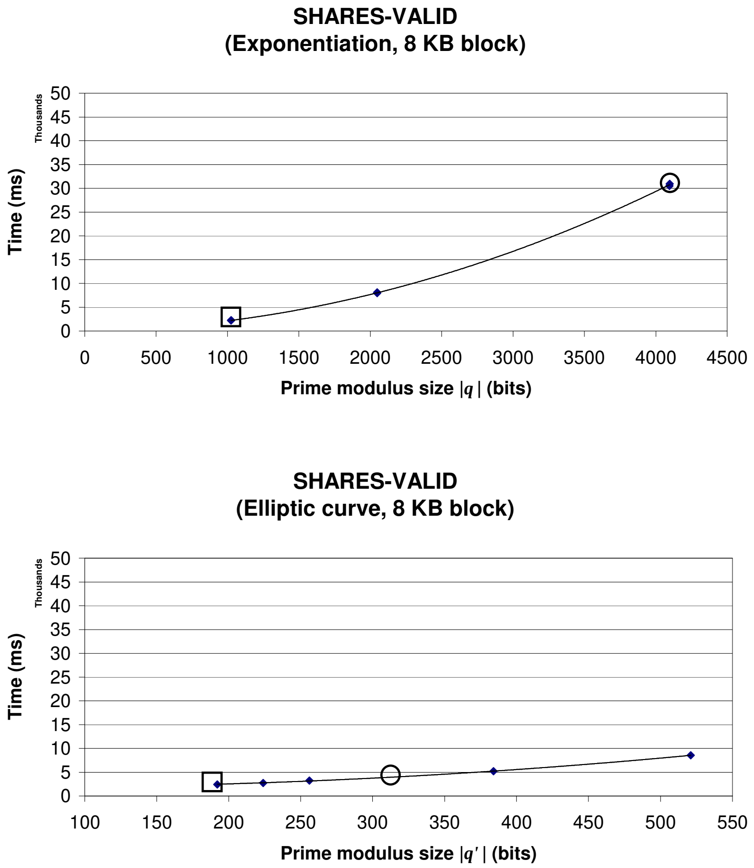

Figure 5.6: Graphs of the time taken by SHARES-VALID with both exponentiation and elliptic curve witness functions, as a function of finite field bit size. The squares and circles highlight points that require roughly equivalent computational effort to solve the DLP. The elliptic curves were computed over the finite fields specified in the Digital Signature Standard [51]. Redistribution was from an (4,8) threshold sharing scheme to an (4,8) scheme.

Operation	REPLICA (ms)	THRESHOLD (ms)	HYBRID (ms)
STORE	31	73	99
REDISTRIBUTE	39	92	153
RETRIEVE	2	3	2

Table 5.4: Time taken to store, redistribute, and retrieve a 0-byte file in Hathor. Each data point represents the average cost over 100 executions of an operation.

DISTRIBUTE operations at the servers. Recalling the relationship between the threshold parameter m and number of shareholders n:

$$m \leq \left\lfloor \frac{n+2}{3} \right\rfloor$$

(Equation (3.18), repeated for convenience), I configured REPLICA, THRESHOLD, and HYBRID as 3-of-7 distribution schemes.

The results for REDISTRIBUTE are for redistribution from a 3-of-7 scheme to a 3-of-7 scheme. For shares of files in THRESHOLD and shares of encryption keys in HYBRID, REDISTRIBUTE includes the execution of the VSR protocol (Figure 3.3). For REPLICA and HYBRID, REDISTRIBUTE includes the time taken for three old servers to send their replicas of a file to the new servers; with three received replicas, the new servers would be able to detect up to two faulty old servers by comparing the replicas.

To emphasize the added costs of inter-server communication and disk accesses over raw computation, I used similar bit sizes for the prime number moduli as for the VSR experiments (Section 5.2). REPLICA and HYBRID use AES encryption in electronic code-book mode with a 256-bit key. THRESHOLD performs threshold sharing modulo a 1024-bit prime number, and witness function exponentiation modulo a 1025-bit prime number. HYBRID performs threshold sharing modulo a 257-bit prime number, and witness function exponentiation modulo a 1025-bit prime number. Because the secret size (*i.e.*, the AES key size) for HYBRID is fixed at 256 bits, one can reduce the cost of witness generation by using the smaller modulus for threshold sharing (and thus reducing the cost of exponentiation (Table 5.1)). The security of the witness values is not affected in practice; recall that the Digital Signature Standard [51] specifies a 160-bit modulus for its exponents.

I ran experiments that measured the time taken to store, redistribute, and retrieve a file of zero bytes in length, in order to understand the overheads introduced by THRESHOLD and HYBRID over the baseline of REPLICA. Table 5.4 shows results of these experiments. For STORE, the 0-byte

Operation	Average overhead (ms)	Standard deviation (ms)
STORE	69	1
REDISTRIBUTE	111	7
RETRIEVE	0	1

Table 5.5: Average STORE, REDISTRIBUTE, and RETRIEVE overhead of HYBRID over REPLICA.

overhead of THRESHOLD and HYBRID over REPLICA arises from the time taken to generate witness (step 2 of INITIAL in the VSR protocol of Figure 3.3). The overhead of THRESHOLD comes from the time taken to compute witnesses for the coefficients of the share generation polynomial, even though these witnesses are not sent to the servers. The overhead of HYBRID comes from the time taken to generate a random AES key and the time taken to generate key shares and the key witness, even though no data is encrypted. For REDISTRIBUTE, the 0-byte overhead of the other schemes over REPLICA is due to the cost of the VSR protocol. The overhead of THRESHOLD comes from the time taken to generate and broadcast SUBSHARES-VALID witnesses (step 2 of REDIST) from old servers to new servers, even though the subshares are of zero size. The overhead of HYBRID comes from the time taken to execute the VSR protocol for shares of the AES key, even though the (encrypted) file is of zero size.

The trend in overheads of THRESHOLD and HYBRID over REPLICA become very different as the file size increases. The graphs in Figure 5.7 show the time taken by STORE, REDISTRIBUTE, and RETRIEVE for different file sizes. The overhead of THRESHOLD over REPLICA for all operations increases as the file size increases, as one would expect given the results in Figure 5.4. On the other hand, the overhead of HYBRID over REPLICA for all operations remains roughly constant; Table 5.5 shows the average overhead for each operation. Interestingly, HYBRID imposes no overhead over REPLICA for RETRIEVE.

5.4 Summary

From the results presented in Section 5.2, we see that the bulk of the cost of the VSR protocol comes from the time taken by verification-related operations: generation of SUBSHARES-VALID witnesses at old shareholders, and verification that the SHARES-VALID and SUBSHARES-VALID conditions hold at the new shareholders. Improvements in CPU performance, or special hardware for arbitrary-precision integer operations, would likely improve the performance of the protocol.

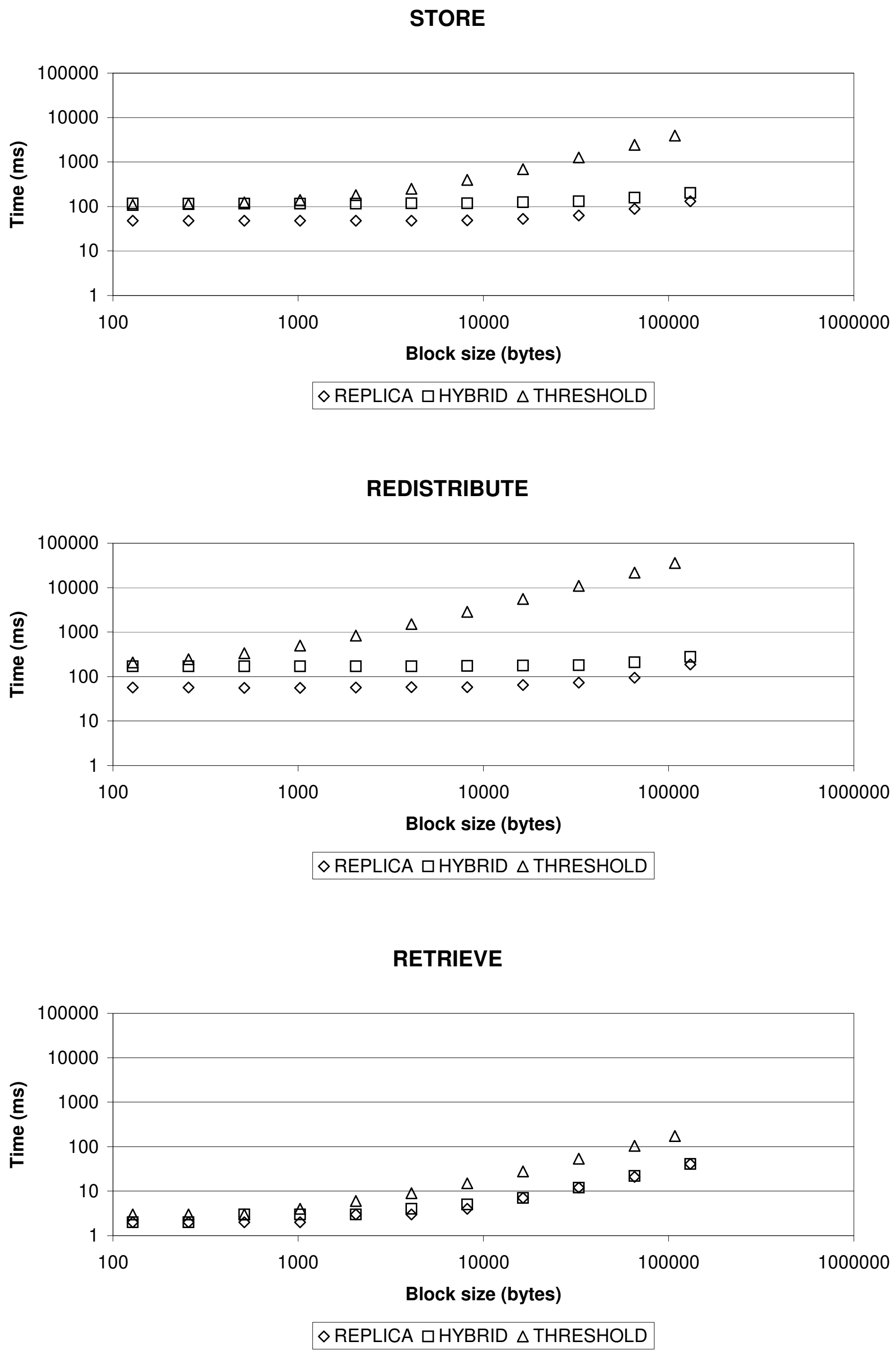

Figure 5.7: Log-scale graphs of the time taken by STORE, REDISTRIBUTE, and RETRIEVE in Hathor, as a function of file size. REPLICA, THRESHOLD, and HYBRID are 3-of-7 distribution schemes. REPLICA and HYBRID use AES encryption in electronic code-book mode with a 256-bit key. The maximum file size for THRESHOLD is 96 KB; above this size, the Ensemble group communication toolkit was unstable. Each data point represents the average cost over 100 executions of an operation.

A surprising result is that the substitution of an elliptic curve-based witness function for an exponentiation-based witness function currently hurts rather than helps performance, though the result makes sense given the analysis in Section 5.2.2. However, we see that as the bit size of underlying fields increases in the future, elliptic curve-based functions will offer better performance than exponentiation-based functions.

The high cost of the VSR protocol, if employed in a storage system, can be offset through careful selection of the data distribution scheme. For a system in which file storage and redistribution operations are infrequent relative to retrieval operations, one may be willing to trade off the slower storage and redistribution performance of HYBRID (compared to REPLICA) in return for the ability to manage the encryption key of a file with its encrypted replicas (Section 4.1), especially given that retrieval operations for HYBRID incur no extra overhead over REPLICA.

Chapter 6

Related Work

And some things that should not have been forgotten were lost. History became legend, legend became myth,...

— Galadriel, in "Lord of the Rings: Fellowship of the Ring" (2001)

My thesis covers both the theoretical and practical aspects of designing a survivable storage system with decentralized recovery. In this chapter, I compare the verifiable secret redistribution (VSR) protocol to other protocols for protecting threshold-shared data in the face of Byzantine adversaries. I then survey survivable storage systems and highlight the similarities and differences between the Hathor system prototype and existing systems. Finally, I discuss some theoretical design studies for survivable storage system architectures.

6.1 Redistribution for threshold sharing schemes

Since the invention of threshold sharing by Blakley [8] and Shamir [49], many researchers have proposed mechanisms to make threshold sharing schemes more *robust*: that is, able to withstand the failure of some of the participants. In this section, I highlight the differences between the VSR protocol and other robust (m,n) threshold sharing schemes, and discuss the relative advantages and disadvantages of the protocol. Table 6.1 summarizes some of the main design points of robust schemes, and shows the greater flexibility offered by the VSR protocol compared to other schemes. As in Chapter 3, I refer to data as secrets, clients as dealers, and servers as shareholders.

Scheme type	*Faulty dealer OK*	*Faulty shareholders OK*	*(m,n) changes OK*
Verifiable secret redistribution	Y	Y	Y
Secret redistribution: Desmedt and Jajodia [16]	N/A	N	Y
Secret redistribution: Frankel *et al.* [18]	Y	Y	N
Proactive secret sharing	Y	Y	N
Verifiable secret sharing	Y	Varies	N/A

Table 6.1: A comparison of robust (m,n) threshold sharing schemes, showing the scheme type, whether the dealer or shareholders may be faulty, and whether m or n may be changed. Note that Desmedt and Jajodia [16] do not specify a phase for the initial distribution of shares of a secret.

My work on the VSR protocol follows from research on robust schemes that involve physical redistribution of shares. Desmedt and Jajodia [16] present a redistribution scheme in which current shareholders distribute subshares of their shares to new shareholders, exactly as in the VSR protocol. However, their scheme only verifies that the SUBSHARES-VALID condition holds, leaving it vulnerable to corruption by faulty shareholders. Frankel *et al.* [18] propose a proactive public-key cryptography scheme in which shareholders redistribute their shares via two steps: a *poly-to-sum* redistribution from an (m,n) polynomial sharing to an (m,m) additive sharing, followed by a *sum-to-poly* redistribution from the additive sharing back to an (m,n) polynomial sharing. In their scheme, the mechanism for the verification of shares after redistribution requires that the membership of the set of shareholders (and thus n) remains static. The distinguishing features of the VSR protocol compared to these schemes include the ability to prevent share corruption by faulty shareholders (up to the limits specified in Section 3.2) and support for changes to m and n.

Other schemes exist that support only limited changes to m or n. Blakley *et al.* consider threshold schemes that *disenroll* (remove) shareholders from the access structure with broadcast messages [7]; the new shareholders are always a subset of the current ones. Cachin proposes a secret sharing scheme that *enrolls* (adds) shareholders in the access structure after the initial sharing [13]; the new shareholders are always a superset of the current ones. By comparison, the VSR protocol supports arbitrary changes to m and n (provided that m is less than or equal to n).

Blundo *et al.* present a scheme in which the dealer broadcasts messages to activate different, possibly disjoint, authorized subsets [10]. A single message activates and deactivates the current subset. All shareholders have a share even if they are not in the active authorized subset, and thus receive a share during the initial distribution of the secret. In contrast, in the VSR protocol only current shareholders have a share, the trade-off being that both current and new shareholders broadcast multiple messages—an important consideration in environments where broadcast is expensive.

The VSR protocol is closely related to *proactive secret sharing* (PSS) schemes, in which shareholders periodically refresh their shares of the secret to counteract a mobile adversary [39]. The key distinction is that PSS schemes keep m and n static, whereas the VSR protocol enables changes to m and n. Herzberg *et al.* [31, 32] present a PSS scheme in which shareholders exchange *update shares* among themselves, and combine the update shares with their current shares. Zhou, Schneider, and van Renesse present a PSS scheme called Asynchronous PSS (APSS), also using update shares, for asynchronous wide-area networks [56, 55]; they independently propose conditions similar to SHARES-VALID and SUBSHARES-VALID to verify the validity of shares after protocol execution. Though APSS does not support changes to m and n, it does have the advantage over the VSR protocol of not requiring a broadcast channel. Frankel *et al.* [19, 20, 21] and Rabin [44] propose several PSS schemes in which shareholders exchange subshares of their shares among themselves, similar to the VSR protocol. Each shareholder then combines the received subshares to generate a new share. As with the proactive cryptography scheme of Frankel *et al.* described above [18], the mechanisms for the verification of new shares require that m and n remain the same.

The VSR protocol is also related to *verifiable secret sharing* (VSS) schemes, which are designed to verify that a dealer distributes valid shares of a secret, and to enable (correct) shareholders to agree that they have valid shares. Non-interactive VSS schemes by Feldman [17] (Section 3.1.3) and Pedersen [41] assume that only the dealer may be faulty. Interactive VSS schemes assume that either the dealer or some of the shareholders may be faulty, and include multiple rounds of communication between the dealer and the shareholders to identify faulty participants; representative examples include schemes by Chor *et al.* [15], Benaloh [5], Gennaro and Micali [25, 26], Goldreich *et al.* [28], and Rabin and Ben-Or [43, 45]. Interactive schemes, at best, only tolerate shareholders that become faulty before or during the initial distribution, while the VSR protocol distinguishes itself in its ability to tolerate shareholders that become faulty after the initial distribution of the secret.

6.2 Survivable storage systems

The Hathor system prototype is an experimental platform on which to evaluate the cost of storage, redistribution, and retrieval for different data distribution schemes. As such, its design involves similar decisions as for survivable storage systems: whether or not to redistribute data in response to server failures, and what type of server failures to tolerate. In this section, I discuss the design points (summarized in Table 6.2) that distinguish Hathor from related systems.

System	*Redistribute data?*	*Server failure mode*	*Distribution schemes*
Hathor	Yes	Byzantine	Encrypted replication Threshold sharing Hybrid sharing + replication
Farsite	Yes	Byzantine	Encrypted replication
PAST	Yes	Byzantine	Encrypted replication
Pond	Yes	Byzantine	Encrypted replication Erasure-resilient coding
InterMemory	Yes	Crash	Erasure-resilient coding
Pangaea	Yes	Crash	Replication
e-Vault	No	Byzantine	Information dispersal
Publius	No	Byzantine	Hybrid sharing + replication
PASIS	No	Crash	Encrypted replication Information dispersal Threshold sharing

Table 6.2: A comparison of survivable storage systems, showing: whether or not redistribution of data on server failure is supported, what type of server failures are tolerated, and what data distribution schemes are implemented.

The underlying research purpose behind Hathor is to develop a decentralized redistribution protocol for threshold-shared data in a system with dynamic membership. As such, it stands in contrast to Farsite [1, 11], PAST [47], and Pond [46] (the OceanStore prototype [37]), which implement redistribution mechanisms for replicated data only (though Farsite also implements a PSS scheme for refreshing signature keys). However, Hathor and these systems do share common design features. They incorporate mechanisms to redistribute data in response to server failures. Also, they are designed under the assumption that servers suffer Byzantine failures, and thus they include mechanisms to verify the validity of data received from servers during I/O operations.

InterMemory [14, 27] is a long-term archival service. It stores data using a scheme based on an erasure-resilient code [9]: one server stores a full copy of the original data, and another n servers store n erasure-coded *fragments* of the data, any $n/2$ of which can be used to reconstruct the data. InterMemory redistributes data when a server fails by reconstructing a full copy or generating a new fragment (depending on what the failed server held) on a replacement server. Unlike Hathor, InterMemory assumes that servers suffer crash failures. Thus, InterMemory does not include mechanisms to verify the validity of data received from servers during I/O operations. In practice, a system designed on the assumption of crash failures may be limited to deployment in closed networks, in which hosts are under the control of a trusted administrator.

Pangaea [48] is a scalable storage utility that uses a high degree of replication to preserve data availability in the face of server failures. It implements a tree-structured file system with *gold* and *bronze* replicas of regular files and directory files: a gold replica of a file points to other gold replicas of the file, to the bronze replicas of the file, and to the gold replicas of the parent and children, while a bronze replica of a file points to the gold replicas of the file and the parent. The distinction between replica types reduces the overhead of managing file replicas, because only gold replicas need to be updated when the directory topology changes, or when the gold replica of a child or parent fails. In particular, the system can quickly create new bronze replicas to replace lost ones. Like InterMemory, Pangaea assumes that servers suffer crash failures, and thus comes with same caveat regarding deployability.

e-Vault [33, 23] and Publius [52] rely upon the inherent redundancy in the supported data distribution schemes to protect the availability of data from server failures, as opposed to redistributing data. e-Vault uses Rabin's information-dispersal algorithm [42] to generate fragments of the (optionally encrypted) data for each server. Publius uses a distribution scheme identical to HYBRID (Section 4.1). For all of these systems, the distribution parameters (the number of servers used to store data, and the number of correct servers required to retrieve the data) are static. Thus, in contrast to Hathor, none of these systems can adapt the parameters in response to either server failures or the addition of new servers (apart from retrieving and re-storing the file).

PASIS [53, 54] also relies upon the inherent redundancy in the supported schemes to protect data. It implements several distribution schemes to evaluate their relative security, performance, and availability characteristics, including encrypted replication, Rabin's IDA, and Shamir's threshold sharing scheme [49]. My research on the VSR protocol and Hathor began as a project to devise decentralized recovery protocols for the distribution schemes in PASIS.

6.3 Design studies for survivable storage systems

In addition to the systems described in Section 6.2, there are several theoretical design studies of survivable storage systems. I present these studies here, and compare them to my work.

Herlihy and Tygar propose two different system designs for making replicated data secure [30]. One design uses symmetric encryption to preserve the confidentiality of replicated data: the system distributes shares of the encryption key with Shamir's (m,n) threshold scheme [49], and retrieves m shares to reconstruct the key prior to performing I/O operations on the data. The other design uses asymmetric encryption: the system distributes shares of a decryption key K_d and an encryption key

K_e with Shamir's (m_d,n) and (m_e,n) respectively. This allows the system to use different threshold values on read (m_d) and write (m_e) operations, and thus allows the end user to tune the performance, availability, and security characteristics of each operation. Unlike Hathor, they do not propose any mechanisms for changing the threshold parameters during system operation (though they do propose a mechanism for changing the key in the private-key system).

Anderson proposes a design for a long-term data repository called Eternity [3]. The primary design goal of Eternity is to protect data against attempts to censor stored data by removal or corruption. Anderson identifies a number of existing technologies that could be used to build Eternity, including protocols for wide-area data replication, encrypted and anonymized communications, and Byzantine agreement. He also identifies open-ended research problems in the design of Eternity, including the need for long-term data indexing services, payment systems, and reliable distributed clocks. My work on the VSR protocol helps to address the primary goal of Eternity, by providing a mechanism to both redistribute data upon server removal and to detect when faulty servers send invalid data.

Alon *et al.* present a distribution scheme to guarantee the availability of data in environments where up to half of the storage servers suffer Byzantine failures [2]. In their scheme, a SHARE function distributes fragments of the data to servers with a Reed-Solomon error-correcting code [24]. Conceptually, the servers are vertices in a *store graph*. SHARE also generates verification information for each vertex, which is used to cross-check both the fragments and the verification information of neighboring vertices. The scheme offers lower space overheads than traditional error-correcting codes, but with the trade-off that (with negligible probability) a corrupted fragment may go undetected during reconstruction of the data. In contrast to my work, Alon *et al.* do not address the problem of guaranteeing the confidentiality of data in the face of Byzantine server failures, considering it to be an orthogonal issue to availability.

Chapter 7

Conclusions and future work

The central thesis of this dissertation is that to truly preserve data for the long term, a survivable storage system must include recovery protocols capable of overcoming server failures, adapting to changing availability or confidentiality requirements, and operating in a decentralized manner. To support this thesis, I presented the design and performance analysis of the verifiable secret redistribution (VSR) protocol. I showed how the VSR protocol is able to preserve the long-term availability and confidentiality of data stored using threshold sharing schemes by implementing the desired capabilities for recovery protocols.

A simple, yet critical, observation underpins the design of the VSR protocol: a more accurate model of a survivable storage system must assume dynamic membership, in which participating servers may join and leave the system. In the limit, a set of new servers after a round of membership changes may be disjoint from the old set. This assumption impacts the design of mechanisms to prevent an adversary from corrupting the execution of recovery protocols. In protocols that assume static membership, participating servers verify correct protocol execution by using information they have received in a previous execution. Under an assumption of dynamic membership, one must include mechanisms in the recovery protocol for new servers to obtain verification information during the current execution, as they will have no information from any previous execution.

The VSR protocol has a high computational cost, but one can mitigate the cost through careful selection of the data distribution scheme in a survivable storage system. For systems in which data storage and redistribution operations are infrequent relative to retrieval operations, a hybrid threshold sharing and encrypted replication scheme—in which shares of encryption keys are stored with encrypted replicas of data—offers the ability to manage encryption keys with data. Such ease of management comes at the cost of a modest slowdown in storage and redistribution performance compared to a standard encrypted replication scheme.

7.1 Research contributions

- I presented a new model of a mobile adversary who subverts servers in a storage system. The new model assumes dynamic membership in the set of participant servers. This assumption imposes a pair of design requirements on any recovery protocol used in the system to counteract the adversary: it must include mechanisms for new servers to obtain the information needed to verify correct protocol execution, and it must allow the system to change the threshold parameters of the underlying data distribution scheme. These requirements are in addition to those that arise under an assumption of static membership in the set of servers.

- I designed and implemented the VSR protocol for the redistribution of shares of a secret originally distributed with Shamir's threshold sharing scheme [49]. The protocol performs redistribution from an authorized subset of old shareholders to all new shareholders. I showed that the protocol satisfies all of the design requirements needed to counteract a mobile adversary in a system with dynamic membership. In particular, I proved that the shares held by shareholders after redistribution can be used to reconstruct the original secret, provided that all of the verification conditions in the protocol hold.

- I conducted a performance analysis to confirm that the bulk of the computational cost of the VSR protocol is in the time taken to compute the witness functions used in verification-related operations. Surprisingly, an elliptic curve-based witness function yields poorer performance than an exponentiation-based witness function, when comparing functions that are currently considered to be secure. I also analyzed the performance trade-offs between different data distribution schemes in an experimental storage system.

7.2 Future work

There are many other open issues in survivable storage systems research to explore beyond that of failure recovery. One area of potential exploration includes developing mechanisms for self-aggregation: can we arrange for a set of connected storage components (bricks or workstations) to discover each other, pool their storage, and present a single system image to the end user? Self-aggregation is easier in systems where physical connectivity implies membership in the set of components (*e.g.*, a rack of disks), but is more difficult in a distributed environment (*e.g.*, a network of workstations: when a new workstation joins the network, should it immediately incorporate itself into the storage system, and should the other components trust it?). How large can the system grow before it requires a dedicated entity to perform management functions, such as partitioning the available capacity across different data and user types? Can the system delegate those functions to one of the storage components, and (especially in the case where a component may be a user's workstation) what impact will that have on the performance of the chosen component?

Failure recovery is far from a solved problem, though. This dissertation investigates the question "what do we do when system components fail", but prompts the question "when have failures occurred". When one moves beyond considering crash failures, and begins to think about compromised components that eavesdrop on data (but appear outwardly to function correctly), it becomes clear that detecting failures is a non-trivial problem. Possible avenues of future research include developing temporal models of how frequently components are likely to become compromised, which can then be used by a system to decide when to execute a recovery protocol proactively.

Appendix A

REDISTRIBUTE for REPLICA and HYBRID

In this appendix, I present the REDISTRIBUTE operation state machines for the REPLICA and HYBRID data distribution schemes in Hathor.

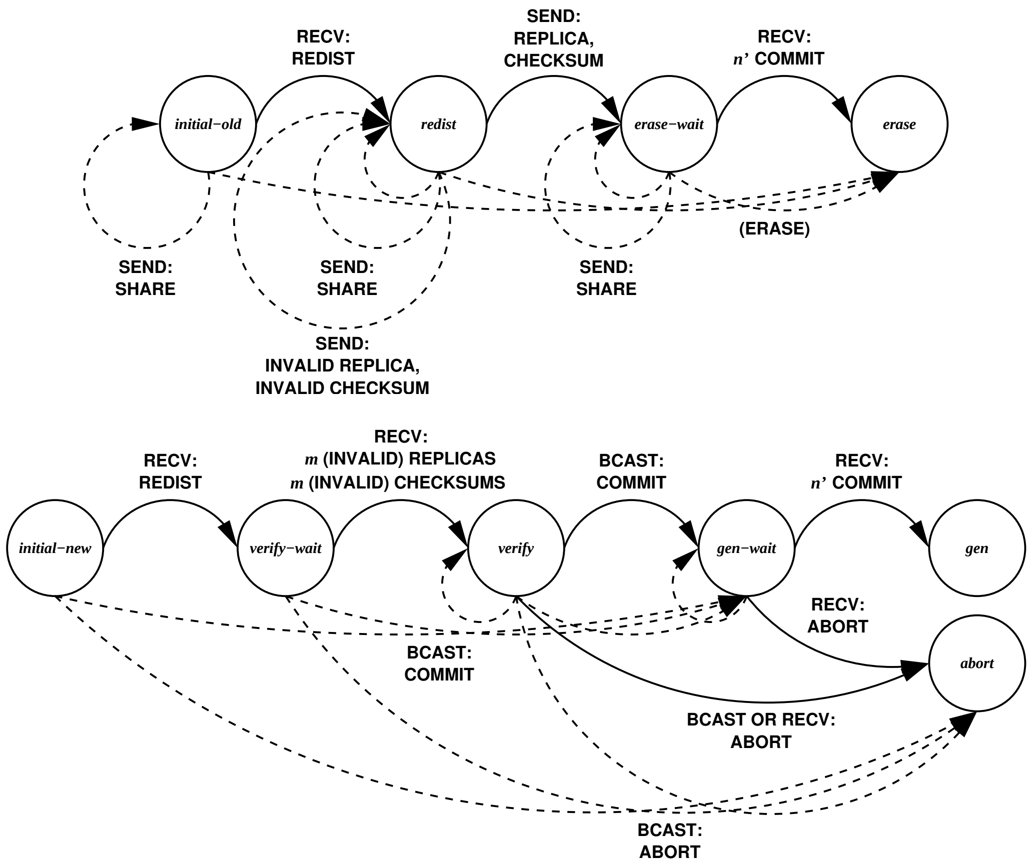

Figure A.1: REDISTRIBUTE state machines for old and new servers, in a system that uses the REPLICA scheme. States correspond to intermediate computations, while state transitions correspond to sent or received messages. Old servers start in the *initial-old* state. New servers start in the *initial-new* state. Solid arrows indicate transitions taken by correct servers, while dashed arrows indicate transitions that may be taken by faulty servers. Unlabelled dashed arrows that loop to the same state indicate no-op self-transitions.

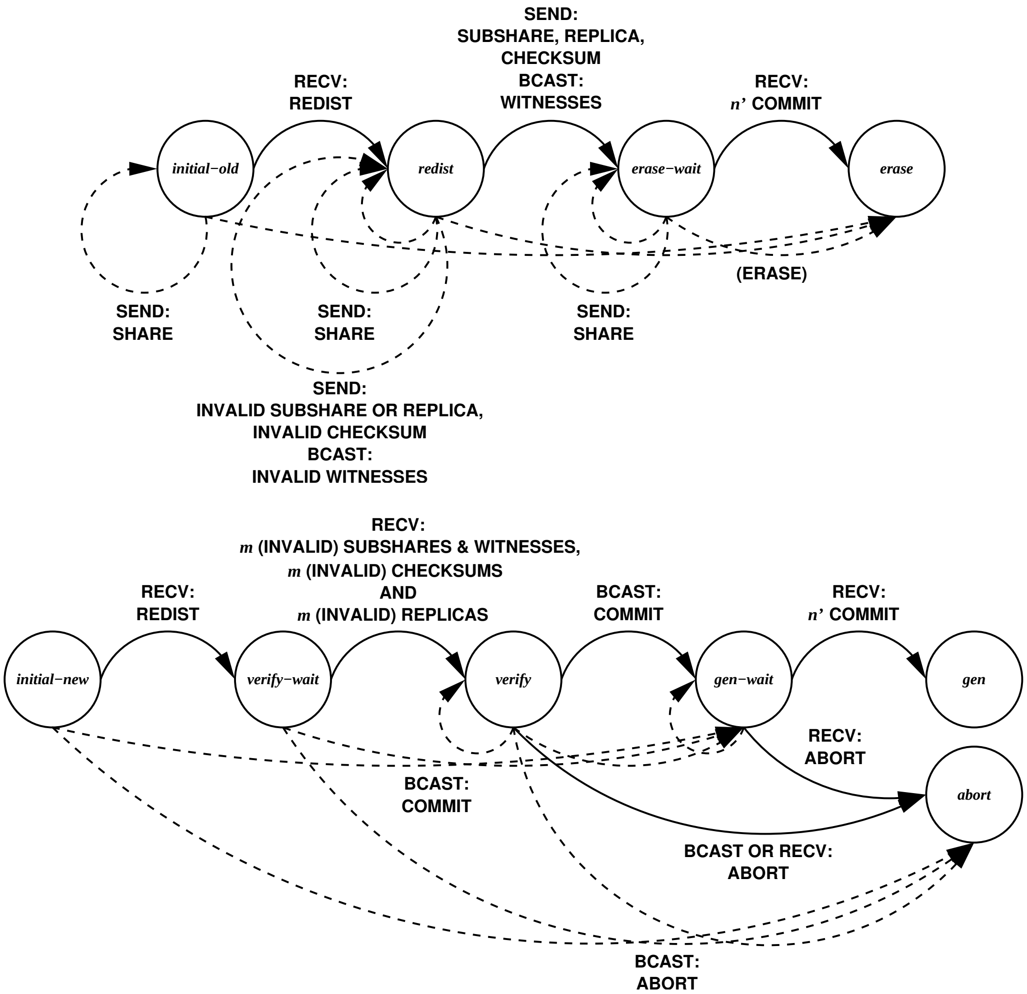

Figure A.2: REDISTRIBUTE state machines for old and new servers, in a system that uses the HYBRID scheme. States correspond to intermediate computations, while state transitions correspond to sent or received messages. Old servers start in the *initial-old* state. New servers start in the *initial-new* state. Solid arrows indicate transitions taken by correct servers, while dashed arrows indicate transitions that may be taken by faulty servers. Unlabelled dashed arrows that loop to the same state indicate no-op self-transitions.

Bibliography

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proc. of the 5th Symp. on Operating Systems Design and Implementation*. Dec. 2002.

[2] N. Alon, H. Kaplan, M. Krivelevich, D. Malkhi, and J. Stern. Scalable secure storage when half the system is faulty. In *Proc. of the 27th Intl. Coll. on Automata, Languages and Programming*, pp 576–587. July 2000.

[3] R. J. Anderson. The Eternity service. In *Proc. of PRAGOCRYPT 1996, Intl. Conf. on the Theory and Applications of Cryptology*, pp 242–253. Oct. 1996.

[4] R. A. Beaumont. *Linear Algebra*. Harcourt, Brace & World, Inc., 1965.

[5] J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Proc. of CRYPTO 1986, the 6th Ann. Intl. Cryptology Conf.*, pp 213–222. 1987.

[6] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, United Kingdom, 1999.

[7] B. Blakley, G. R. Blakley, A. H. Chan, and J. L. Massey. Threshold schemes with disenrollment. In *Proc. of CRYPTO 1992, the 12th Ann. Intl. Cryptology Conf.*, pp 540–548. Aug. 1992.

[8] G. R. Blakley. Safeguarding cryptographic keys. In *Proc. of the Natl. Computer Conf.*, 1979.

[9] J. Blömer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-based erasure-resilient coding scheme. Technical Report TR-95-048, Intl. Computer Science Inst., UC Berkeley, Berkeley, CA, Aug. 1995.

[10] C. Blundo, A. Cresti, A. D. Santis, and U. Vaccaro. Fully dynamic secret sharing schemes. *Theoretical Comput. Sci.*, 165(2):407–440, Oct. 1996.

[11] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proc. of SIGMETRICS 2000, the Intl. Conf. on Measurement and Modeling of Computing Systems*, pp 34–43. June 2000.

[12] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation (Extended abstract). In *Proc. of CRYPTO 1992, the 12th Ann. Intl. Cryptology Conf.*, pp 200–207. Aug. 1992.

[13] C. Cachin. On-line secret sharing. In *Proc. of the 5th IMA Conf. on Cryptography and Coding*, pp 90–198. Dec. 1995.

[14] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos. A prototype implementation of archival intermemory. In *Proc. of the 4th ACM Intl. Conf. on Digital Libraries*, pp 28–37. Aug. 1999.

[15] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (Extended abstract). In *Proc. of the 26th IEEE Ann. Symp. on Foundations of Computer Science*, pp 383–395. Oct. 1985.

[16] 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.

[17] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. of the 28th IEEE Ann. Symp. on Foundations of Computer Science*, pp 427–437. Oct. 1987.

[18] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *Proc. of the 38th IEEE Ann. Symp. on Foundations of Computer Science*, pp 384–393. Oct. 1997.

[19] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Proactive RSA. In *Proc. of CRYPTO 1997, the 17th Ann. Intl. Cryptology Conf.*, pp 440–454. Aug. 1997.

[20] Y. Frankel, P. D. MacKenzie, and M. Yung. Adaptively-secure optimal-resilience proactive RSA. In *Proc. of ASIACRYPT1999, the 5th Intl. Conf. on the Theory and Application of Cryptology and Information Security*, pp 180–194. Nov. 1999.

[21] Y. Frankel, P. D. MacKenzie, and M. Yung. Adaptive security for the additive-sharing based proactive RSA. In *Proc. of PKC 2001, the 4th Intl. Workshop on Practice and Theory in Public Key Cryptography*, pp 240–263. Febraury 2001.

[22] S. Frølund, A. Merchant, Y. Saito, S. Spence, and A. Veitch. FAB: enterprise storage systems on a shoestring. In *Proc. of the 9th IEEE Workshop on Hot Topics in Operating Systems*. May 2003.

[23] J. A. Garay, R. Gennaro, C. S. Jutla, and T. Rabin. Secure distributed storage and retrieval. *Theoretical Comput. Sci.*, 243(1–2):363–389, July 2000.

[24] P. Gemmell and M. Sudan. Highly resilient correctors for polynomials. *Inf. Process. Lett.*, 43(4):169–174, Sept. 1992.

[25] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Proc. of EUROCRYPT 1996, the Intl. Conf. on the Theory and Application of Cryptographic Techniques*, pp 354–371. May 1996.

[26] R. Gennaro and S. Micali. Verifiable secret sharing as secure computation. In *Proc. of EUROCRYPT 1995, the Intl. Conf. on the Theory and Application of Cryptographic Techniques*, pp 168–182. May 1995.

[27] A. V. Goldberg and P. N. Yianilos. Towards an archival Intermemory. In *Proc. of the IEEE Forum on Reasearch and Technology Advances in Digital Libraries*, pp 147–156. Apr. 1998.

[28] O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptograhpic protocol design. In *Proc. of CRYPTO 1986, the 6th Ann. Intl. Cryptology Conf.*, pp 171–185. 1987.

[29] M. Hayden and R. van Renesse. Optimizing layered communications protocols. In *Proc. of the 6th IEEE Symp. on High Performance Distributed Computing*, Aug. 1997.

[30] M. Herlihy and J. D. Tygar. How to make replicated data secure. Technical Report CMU-CS-87-143, Sch. of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, Aug. 1987.

[31] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Proc. of the 4th ACM Intl. Conf. on Computer and Communications Security*, pp 100–110. Apr. 1997.

[32] A. Herzberg, S. Jarekci, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Proc. of CRYPTO 1995, the 15th Ann. Intl. Cryptology Conf.*, pp 339–352. Aug. 1995.

[33] A. Iyengar, R. Cahn, J. A. Garay, and C. Jutla. Design and implementation of a secure distributed data repository. In *Proc. of IFIP/SEC 1998, the 14th Ann. Intl. Conf. on Information Security*. Sept. 1998.

[34] S. Kirkpatrick, W. Wilcke, R. Garner, and H. Huels. Percolation in dense storage arrays. *Physica A*, 314(1–4):220–229, Nov. 2002.

[35] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(117):203–209, Jan. 1987.

[36] A. I. Kostrikin. *Introduction to Algebra*. Springer-Verlag, 1982.

[37] J. Kubiatowicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherspoon, and B. Zhao. OceanStore: An architecture for global-state persistent storage. In *Proc. of ASPLOS IX, the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp 190–201, Nov. 2000.

[38] V. S. Miller. Use of elliptic curves in cryptography. In *Proc. of CRYPTO 1985, the 5th Ann. Intl. Cryptology Conf.*, pp 417–426. Aug. 1986.

[39] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. of the 10th Ann. ACM Symp. on Principles of Distributed Computing*, pp 51–59. Aug. 1991.

[40] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. of 1988 ACM SIGMOD Intl. Conf. on Management of Data*, pp 109–116, June 1988.

[41] T. P. Pedersen. Non-iteractive and information-theoretic secure verifiable secret sharing. In *Proc. of CRYPTO 1991, the 11th Ann. Intl. Cryptology Conf.*, pp 129–140. Aug. 1991.

[42] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, Apr. 1989.

[43] T. Rabin. Robust sharing of secrets when the dealer is honest or cheating. *J. ACM*, 41(6):1089–1109, Nov. 1994.

[44] T. Rabin. A simplified approach to threshold and proactive RSA. In *Proc. of CRYPTO 1998, the 18th Ann. Intl. Cryptology Conf.*, pp 89–104. Aug. 1998.

[45] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. of the 21st Symp. on the Theory of Computing*, pp 73–85. May 1989.

[46] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The OceanStore prototype. In *Proc. of the 2nd Conf. on File and Storage Technology*. Mar.–Apr. 2003.

[47] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. of the 18th Symp. on Operating Systems Principles*, pp 188–201. Oct. 2001.

[48] Y. Saito, C. Karamonolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the Pangaea wide-area file system. In *Proc. of the 5th Symp. on Operating Systems Design and Implementation*. Dec. 2002.

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

[50] Shamus Software Ltd., Ballybough, Dublin 3, Ireland. *M.I.R.A.C.L. Users Manual*, Nov. 2002.

[51] US National Institute of Standards and Technology. Digital signature standard (DSS). FIPS PUB 186-2, Jan. 2000. Includes change notice.

[52] M. Waldman, A. D. Rubin, and L. F. Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. of the 9th USENIX Security Symp.*, pp 59–72. Aug. 2000.

[53] J. J. Wylie, M. Bakkaloglu, V. Pandurangan, M. W. Bigrigg, S. Oguz, K. Tew, C. Williams, G. R. Ganger, and P. K. Khosla. Selecting the right data distribution scheme for a survivable storage system. Tech. Rep. CMU-CS-01-120, Sch. of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, May 2001.

[54] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliççöte, and P. K. Khosla. Survivable information storage systems. *IEEE Computer*, pp 61–68, Aug. 2000.

[55] L. Zhou, F. B. Schneider, and R. van Renesse. APSS: Proactive secret sharing in asynchronous systems. In preparation.

[56] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. *ACM Trans. Comput. Syst.*, 20(4):329–368, Nov. 2002.