IMPLEMENTATION AND EVALUATION OF A STORAGE SYSTEM FOR LONG TERM DATA

A Thesis in
Computer Science and Engineering
by Dharani Sankar Vijayakumar

copyright 2008 Dharani Sankar Vijayakumar

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science

December 2008
The thesis of Dharani Sankar Vijayakumar was reviewed and approved* by the following:

Bhuvan Urgaonkar  
Assistant Professor of Computer Science and Engineering  
Thesis Advisor

Trent Jaeger  
Associate Professor of Computer Science and Engineering

Raj Acharya  
Professor of Computer Science and Engineering  
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.
ABSTRACT

Many factors such as legislations, family heritage necessitates data such as medical records, legal documents to be secured and preserved for a longer period of time. The unique security requirement of long term data is to provide information theoretical confidentiality guarantee for the data stored on the servers. The information theoretical confidentiality property defies any cryptanalysis attack on the encoded data stored on the storage servers. Another important requirement of a long-term storage system is to ensure the availability of any stored data whenever requested for reading. In this work, we propose and evaluate our techniques to maintain the unique data confidentiality and the availability requirements of a long term storage system designed using an appropriate encoding technique called secret sharing. This study exposes the interesting tradeoffs that exist between system performance, security and availability of a secret sharing based long term storage system.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>viii</td>
</tr>
</tbody>
</table>

**Chapter 1** Introduction ................................................................. 1

1.1 Secret Sharing ................................................................. 1
1.2 System Model ................................................................. 2
1.3 Encoding and Decoding performance of Secret Sharing ..................... 4
  1.3.1 Encoding ................................................................. 4
  1.3.2 Decoding ................................................................. 6
1.4 Availability offered by secret sharing based systems ....................... 7
1.5 Problem Definition ............................................................. 7
1.6 Organization ................................................................. 10

**Chapter 2** Related Work ............................................................... 11

2.1 Proactive Secret Sharing (PSS) .................................................. 11
  2.1.1 Problems using Proactive Secret Sharing (PSS) for archival system ... 13
2.2 Jajodia’s protocol ............................................................... 13
  2.2.1 Problems using Jajodia’s protocol ..................................... 14

**Chapter 3** Operating Algorithms ..................................................... 16

3.1 Unishare Splitting ............................................................... 17
3.2 Unishare Splitting’s fault tolerance evaluation .............................. 18
  3.2.1 Characterizing Fault tolerance of Unishare split objects .......... 19
  3.2.2 Unishare split’s availability for real systems ...................... 21
3.3 Multishare Splitting algorithm ............................................... 23

**Chapter 4** Implementation and Evaluation of redistribution schemes ....... 26
4.1 Implementation Details .......................................................... 26
  4.1.1 Metadata Manager ......................................................... 27
  4.1.2 Redistribution Service .................................................. 28
4.2 Evaluation ................................................................. 31
Chapter 5 Conclusions ................................................................. 33

Bibliography ................................................................................. 34
LIST OF FIGURES

Figure 1: System model ................................................................. 2

Figure 2: Herzberg protocol for secret redistribution from (8, 11) to (9, 11) ............ 12

Figure 3: Jajodia protocol for secret redistribution from (8, 11) to (9, 11) ............ 14

Figure 4: Unishare splitting protocol for secret redistribution from (8, 11) ............ 16

Figure 5: Unishare fault tolerance illustration using (9, 11) configuration ............. 19

Figure 6: Availability after redistribution from (10, 20) to (11, 20) using Unishare splitting ................................................................. 22

Figure 7: Availability after redistribution from (11, 20) to (12, 20) using Unishare splitting ................................................................. 22

Figure 8: Multishare splitting protocol for secret redistribution from (8, 11) ............ 24

Figure 9: Creation of three secondary shares from two primary shares using Multishare splitting ................................................................. 25

Figure 10: Metadata maintenance in prototype deployment ..................................... 27

Figure 11: Redistribution participation scenario ............................................. 28

Figure 12: Unishare redistribution from (6, 11) to (7, 11) for a data object ............. 29

Figure 13: Multishare redistribution from (6, 11) to (7, 11) for a data object .......... 30

Figure 14: Unishare Vs Multishare completion times for data objects of size 100KB and striped using (6, 11) secret sharing configuration ............................................. 31
LIST OF TABLES

Table 1 Computation times for secret sharing encoding ........................................... 4

Table 2 Computation times for secret sharing decoding ............................................. 6
ACKNOWLEDGEMENTS

I would like to thank Dr. Anand Sivasubramiam and Dr. Bhuvan Urgaonkar for guiding me right from the beginning of my graduate studies. I would like to thank Dr. Trent Jaeger for being part of my thesis committee. Special thanks to Shiva, Niranjan, Sriram and my family members for all the inspiration during the challenging times at Penn State. I would like to thank all Computer Systems Lab members for providing an amazing environment to research and have fun.
Chapter 1

Introduction

1.1 Secret Sharing

Legislations [Act], family heritage mandates data such as medical records, legal documents to be secured for a longer period of time. Since, such data have meaning for a very long time, using traditional encryption standards like AES and DES to encode this long term data would make the storage servers susceptible to cryptanalysis attacks. On the other hand, secret sharing is a technique that suits the needs of such a long term data due to its information theoretical security property. By guaranteeing information theoretical security, secret sharing makes an attacker gain no meaning from the encoded data if malicious cryptanalysis were to be done. In addition to guaranteeing information theoretical security, secret sharing schemes also offer redundancy and hence can be used to make data available for a longer period of time in spite of storage server failures.

Typically, a secret sharing scheme is defined by two parameters m and n. A (m, n) secret sharing scheme creates n fragments from a data object in such a manner that no fewer than m fragments can be used to reconstruct the data item. Given any m fragments, one can reconstruct the data item but fewer than m fragments will provide zero information about the data item. There are a variety of perfect secret sharing techniques proposed in the literature. We use Shamir secret sharing [Shamir] for the current study. Using Shamir’s secret sharing, any (m, n) secret share configuration can be created using a polynomial P(x) of degree m-1 whose P(0) represents the secret. Hence theoretically, at least m out of n values P(1), P(2), P(3), ..., P(n) are required to reconstruct the secret P(0). This guarantees the information theoretic security of the Shamir’s scheme. Since
any m out of n values P(1), P(2), P(3), …, P(n) are enough to reconstruct P(0), inherent redundancy is guaranteed by Shamir’s scheme.

1.2 System Model

Storage systems such as [Potshards, Tedwong, Pasis] have used secret sharing techniques to encode and decode data to be stored on servers. We consider a similar architecture for our long term storage system. To store long term data which is usually archival (write once and read rarely) in nature, we consider a distributed storage system model where the individual storage nodes are independent of each other in the best case. The archival storage manager which is usually part of the client continually streams data from the clients to the storage servers on the network. The archival storage manager provides an interface to the clients wherein the data is written in the form of archival data objects. Each data object is analogous to a file in a server file system or it could refer to a smaller granularity unit like a disk block or an extent of blocks.

Figure 1 System model
Archival storage managers that deploy secret sharing are equipped with a splitting service that splits each archival object created by a client application into secret shares before sending across the network. In the scenario shown in Figure 1, the two data objects created by two different clients are split using (2, 3) and stored on the servers. For wide area networks, it is usually the case that the storage nodes across sites are independent of each other in terms of security breaches and failure. Therefore, the best way of assigning fragments (or secret shares) of a data object to a set of nodes is one to one, i.e. each fragment goes to a different node. This keeps each data fragment independent and best utilizes the properties of secret sharing schemes and hence m different authentication services have to be breached to expose a data object striped using (m, n) scheme. Typically the value of n used for secret sharing is relatively small (range is 5 to 20) compared to the number of storage nodes N (usually in the range 100 to 1000) in the wide area network. Thus, a load balancer service can be used as shown in Figure 1 to store the secret shares to maintain optimal distributed system utilization. The parameters m and n for secret sharing are dictated by system requirements. For example, if it is known during system design that m or more nodes being compromised and more than k nodes becoming unavailable simultaneously is a possibility the archival storage manager can be tuned to split data objects using a (m+k, n). This, however, means that the total storage space required at the storage nodes is m+ k times that of the actual data generated by the archival applications. Another important parameter is the system performance. The key factors affecting the performance are (a) Encoding/Decoding for secret sharing (b) Network latency and bandwidth (c) Storage access latency. The experiments reveal that the encoding and decoding process is more likely to be the bottleneck for a secret sharing
based system when compared to realistic network bandwidth measured using test bed like planetlab [Planetlab] and practical disk systems [Seagatereport].

1.3 Encoding and decoding performance of secret sharing

1.3.1 Encoding

Secret sharing suffers performance loss due to the CPU computation and random block generation while encoding the data. The below are the encoding bandwidth for secret sharing scheme implementation [SSlibrary] measured using a Intel(R) Xeon(TM) CPU 3.06GHz with physical memory 1GB RAM and cache size of 512 KB (L2 cache).

<table>
<thead>
<tr>
<th>Threshold (m)</th>
<th>Fragment (n)</th>
<th>Bandwidth in KB/sec for 8 byte field</th>
<th>Bandwidth in KB/sec for 64 byte field</th>
<th>Bandwidth in KB/sec for 128 byte field</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>409</td>
<td>524</td>
<td>353</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>334</td>
<td>364</td>
<td>228</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>221</td>
<td>200</td>
<td>132</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>245</td>
<td>233</td>
<td>146</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>149</td>
<td>118</td>
<td>77</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>104</td>
<td>80</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>82</td>
<td>59</td>
<td>38</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>16</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td>11</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>8</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>12</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1: CPU bandwidth of different m, n parameters for a Shamir secret sharing encoding implementation
From Table 1 two important observations can be made,

1. **Field size is an important parameter that can be tuned for performance.**
   Bandwidth doubles by decreasing the field size from 128 bytes to 8 bytes for most (m, n) configurations.

2. **The lower CPU bandwidth values for secret sharing encoding shows us that computation would be the bottleneck when compared to the network or a storage array which are also part of our system model shown in Figure 1.**

The assumption made in Table 1 is,

1. All the experiments in Table 1 ignored the amount of time required to generate random blocks. To encode using (m, n) secret sharing would require a random block of size (n-1) times the data block size. When we experimented using a character device file /dev/urandom to generate huge amount of randomness required for secret sharing encoding the bandwidth went down slightly for all (m,n) configurations (1.93 Kbytes/sec instead of 2 Kbytes/sec for 20,20 scheme). But, since /dev/urandom is not the most secure way of generating random blocks, we do not suggest using /dev/urandom as a random source in real deployments. The archival system would preferably use a high performance hardware random number generator to avoid further performance loss in the encoding stage. The evaluation of time taken to create random blocks using real deployment hardware number generators is beyond the scope of this work.
1.3.2 Decoding

Secret sharing suffers from bandwidth loss when the client computes the original data after reading the individual fragments from the servers on the network as shown in Figure 1. Even though reads are rare unlike continuous writing in the case of an archival storage system; the decoding overhead determines how “thin” (like a mobile device) the archival data readers can be. Table 2 shows the computation time required in the decoding process.

<table>
<thead>
<tr>
<th>Threshold (m)</th>
<th>Fragment (n)</th>
<th>Bandwidth in KB/sec for 8 byte field</th>
<th>Bandwidth in KB/sec for 64 byte field</th>
<th>Bandwidth in KB/sec for 128 byte field</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>160.4</td>
<td>123.4</td>
<td>117.9</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>160.9</td>
<td>121.8</td>
<td>118.5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>66.9</td>
<td>53.2</td>
<td>43.2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>160.1</td>
<td>127.6</td>
<td>117.6</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>68.2</td>
<td>53.3</td>
<td>41.7</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>34.8</td>
<td>26.2</td>
<td>22.2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>19.8</td>
<td>14.8</td>
<td>11.6</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>5.21</td>
<td>4</td>
<td>3.2</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td>1.9</td>
<td>1.6</td>
<td>1.2</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>0.70</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>5.21</td>
<td>4.1</td>
<td>3.1</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>0.9</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>0.3</td>
<td>0.2</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 2 CPU bandwidth of different m, n parameters for a Shamir secret sharing encoding implementation
The decoding bandwidth for secret sharing schemes is even lesser when compared to the encoding bandwidth. Also, the decoding process doesn’t require any random number generation unlike the encoding stage.

1.4 Availability offered by secret sharing based systems

Servers used to store secret sharing fragments can become unreachable due to both transient and permanent failures of intermediate links and the end servers themselves. Availability of a data object refers to the probability that the object can be retrieved on reader request. There have been studies in the past [Sumannath] which found the failure characteristics of Internet based systems like well known webservers and testbed networks like Planetlab [Planetlab]. Using the derived failure characteristics these previous works suggest ways of finding an appropriate value for the (m,n) to achieve certain levels of availability during the system operation. These same techniques can be used to tune the system availability for an archival storage system using secret sharing. But (m,n) values optimized for availability would have to be discarded if the security guaranties provided by the values m and n are not suitable.

1.5 Problem Definition

Even though the read/write performance of secret sharing schemes is a bottleneck, the availability and security guarantees provided by the parameters m and n makes secret sharing a good choice for a secure archival storage design. This thesis looks deeper into the problem of redistribution [Herzberg, Jajodia] in secret sharing schemes used for a secure archival storage system. As mentioned earlier, a (m, n) secret sharing scheme
offers two dimensions, secrecy and fault tolerance. The preserved data item is considered to be safe as long as less than \( m \) fragments are leaked. Hereafter, we will use the term security threshold to denote the minimal number of fragments that need to be compromised to constitute an attack in secret sharing schemes. For \((m, n)\) configurations, the security threshold is equal to \( m \). The preserved data item is available under the loss of at most \( n - m \) fragments. After the archival objects have been originally written onto the storage nodes with a \((m, n)\) secret sharing configuration, intermediate changes to the configuration before the eventual retrieval of data are necessitated by one or combination of the following scenarios.

**Node Compromise**

The scenario where one of the nodes in the wide area network has suffered a security breach resulting in the potential exposure of all the data stored at the node is a possibility. This means that all the archival objects whose secret sharing fragments were stored at the node have been effectively rendered a decrease in security threshold by one to \((m - 1, n)\). This is because an adversary now potentially requires only \( m - 1 \) additional fragments to constitute an attack. To counter this, the administrator can initiate a reconfiguration of the remaining fragments of the affected objects and increase their security threshold by one. In general, a compromise of \( k \) nodes might require a reconfiguration of a rendered \((m - k, n - k)\) to \((m, n - k)\). The archival data objects have sensitive real world meaning over longer periods of time and hence the administrator cannot wait for many such \( k \) nodes to get compromised and eventually making the archival object to appear in clear. We argue
the pros and cons of existing solutions and propose our techniques to solve this problem in later sections.

**Data loss**

Permanent loss of data at a node reduces the configuration of all affected objects to \((m, n - 1)\). The archival manager might wish to use some other node or alternate storage space at the affected node. In either case, a reconfiguration from \((m, n - 1)\) to \((m, n)\) for all affected objects is required.

**Node Expansion**

Due to the lengthy periods of time for which the archival data are stored at the nodes, the data will witness plenty of changes to the underlying storage infrastructure. Storage systems retire slowly compared to the rate of expansion due to the decreasing costs of storage and increasing storage densities over time. Also, another example of node expansion is when the compromised nodes are security-patched and later join the network again. As a result, the stored archival objects may need to be moved to a different configuration to accommodate the new infrastructure. This might mean changes to \(m\) or \(n\) or both.

This thesis doesn’t argue about effective solutions for data loss and node expansion any further. A comprehensive solution for data loss and node expansion will be part of the future work.
1.6 Organization

The rest of this thesis is organized as follows. Chapter 2 presents the relevant related work and the unsolved problems. Chapter 3 proposes the algorithms Unishare splitting and Multishare splitting that can be used to increase the threshold of a secret sharing scheme after node compromise. Chapter 4 presents the implementation and evaluation of the secret redistribution algorithms. Finally, Chapter 5 concludes the thesis with directions on our future work.
Chapter 2

Related Work

Prior research has proposed solutions to restore the security threshold after a security compromise in a secret share holding server. The solutions of interest to us restore the security threshold by changing a (m, n) scheme to (m+k, n) secret sharing configuration without any reconstruction of the original data at the administrator. Any reconstruction effort at the central administrator would make the administrator a single point of data exposure. Hence studies including this do not consider the reconstruction approach to avoid the potential problems due to a careless administrator system which might have malicious insiders physically working along with him or virtually in the form of computer viruses.

2.1 Proactive Secret Sharing (PSS)

PSS is a technique that can be used to sustain the secrecy of the distributed system after node compromises. Using Proactive secret sharing (PSS) a new set of shares can be created by updating the existing shares on the data servers and hence the updated shares can be made to represent a different secret sharing configuration. For Shamir’s scheme this can be done by adding a random polynomial who’s co-efficient is zero as proposed by [Herzberg]. Thus, by adding a random polynomial of degree m+k-1 to an existing secret represented using a polynomial of degree m-1, one can increase the security threshold m to m+k of an (m, n) system.
Figure 2, shows the application of Herzberg protocol to move the configuration of a (8, 11) system to a (9, 11) system by adding a random polynomial $Q(x)$ to the secret representing polynomial $P(x)$. In this example, the original secret represented by the polynomial $P(x)$ is $P(0)$. The degree of the original polynomial $P(x)$ is 7. The newly added random polynomial $Q(x)$ is of degree 8 and has a $Q(0)$ value of 0. The secret $P(0)$ is preserved in the newly formed polynomial $P(x) + Q(x)$ since $Q(0)$ is 0 (and hence $(P+Q)(0)=P(0)$).

**Before redistribution**

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
<th>P11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P1</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Q1</td>
<td></td>
</tr>
</tbody>
</table>

**After redistribution**

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
<th>P11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P1</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Q2</td>
<td></td>
</tr>
</tbody>
</table>

Shares P1, P2, …P11 were created using a polynomial $P(x)$ of degree 7

Share Q1, Q2, … Q11 are created using a polynomial $Q(x)$ of degree 8 and $Q(0)=0$
2.1.1 Problems using PSS in [Herzberg] for archival system

**Nodes churning during reconfiguration**

The [Herzberg] protocol requires the coordinator to update at least \( m \) fragments out of the total \( n \) fragments in order to start increasing the security when \( (m > n/2) \). Thus, for values of \( m \) closer to \( n \), reconfiguration is impossible during periods when the data objects are intermittently unavailable from the coordinator site due to link or machine failures.

2.2 Jajodia’s Protocol

[Jajodia] proposed a generic protocol to move from any \((m, n)\) scheme to a \((m', n')\) scheme. Thus this protocol can be applied to both increase from \( m \) to \( m+k \) after a permanent node compromise, decrease from \( m+k \) to \( m \) after a patch has been applied to the \( k \) earlier compromised machines.
In Figure 3, the first phase is the splitting phase where 11 secret shares each for any 8 out of the 11 primary shares (P1 to P8 in the example shown in Figure 3) are formed by performing a (9, 11) Shamir split. The second phase is the combination phase where the sub blocks from different original shares are combined using (8, 8) interpolation which is the same technique used to create a secret back from 8 fragments.

2.2.1 Problems using Jajodia’s protocol for archival system

We find two problems in using the [Jajodia] technique for moving from (m, n) configuration to (m+k, n) configuration in an archival storage system.
**Slower Completion time**

With large number of objects, performing an $O(n^2)$ communication and computation for reconfiguration might bottleneck the system and disturb the foreground archiving traffic. This is further compounded by the fact that reconfigurations are typically done as batch jobs for millions of objects at a time. This is because they are triggered by node compromise/failure events causing all objects in the node to be reconfigured. As a result, achieving fast completion of reconfiguration tasks becomes a harder goal. Security restoration is a first order requirement of the system as soon as vulnerabilities have been suspected by the administrator. Thus, it is always desirable to reduce the traffic generated and computation required for each data object’s security restoration so that faster completion times of the restoration protocol is highly probable.

**Nodes churning during reconfiguration**

At the time of reconfiguration, a subset of n nodes containing the shares may not be available. In most distributed systems spread over wide geographic sites, nodes experience temporary failures because of reasons like link failure, remote maintenance etc. There is usually a default churning of nodes, i.e. nodes go down temporarily and join the network again at a later time. It is desirable that the reconfiguration protocol proceeds with the alive subset of nodes (with subset of shares < m) instead of waiting an indefinite amount of time for m nodes to be available.
Chapter 3

Operating Algorithms

In this chapter we propose our algorithms which do not suffer from the problems that were earlier identified for [Jajodia] and [Herzberg] schemes.

3.1 Unishare Splitting

Figure 4 – Unishare splitting protocol for secret redistribution from (8, 11) to (9, 11)

We propose a redistribution protocol, the Unishare splitting protocol that can be used when the security threshold of a (m, n) Shamir configuration needs to be increased by k with access to atmost L original shares. The basic idea is that if exactly m − 1 original shares were to be left intact, then the L primary shares should be replaced by a set of secondary shares in such a manner that it must require at least k + 1 of the secondary shares to construct any one of the replaced primary shares. Under the assumption that m
> n/2, this will ensure that the minimal set for secret reconstruction in the resultant
configuration is the set of primary shares (m− 1) combined with (k + 1) secondary shares.
In the Unishare split method, each primary share in the subset v generates t new shares.
The resulting v*t secondary shares replace the L primary shares. When the integer value t
does not wholly divide L, we cannot apply (t, t) splitting to each primary share in the
subset v. In this boundary case, we apply a (t,t+delta) Shamir splitting to the last primary
share in L where delta is a number less than t. This Unishare splitting algorithm was
designed to maximize the number of primary shares getting used (L) and hence to offer
better fault tolerance.

Figure 4 illustrates how the security threshold of a (8, 11) Shamir configuration can be
increased from 8 to 9. This is achieved by using (2, 2) splitting as follows. From the
eleven shares in the (8, 11) initial configuration, seven shares (P1 to P7 in this example)
are left unmodified. Two primary shares P8 and P9 are chosen for (2, 2) splitting
resulting in secondary shares S1 to S4. Finally, the four primary shares P8 to P11 are
securely deleted and replaced by the secondary shares. It can be seen here that an
adversary now requires a minimum of 9 shares (the minimal sets are \{P1, P2, P3, P4, P5,
P6, P7, S1, S2\} and \{P1, P2, P3, P4, P5, P6, P7, S3, S4\}) to retrieve data. On the flip
side, the resultant configuration is inferior to the corresponding (9, 11) Shamir
configuration with respect to fault tolerance. It is important to study this fault tolerance
because it affects the,

- Number of legitimate read requests getting satisfied when the object is in a split
  configuration
• Also, the easiness of moving back from \((m+k, n)\) to \((m, n)\) again depends on the availability of the split object. Now consider the \((9, 11)\) scheme shown in figure. Share \(P8\) can be retrieved back by combining \(S1\) and \(S2\). Similarly, share \(P9\) can be retrieved back by combining \(S3\) and \(S4\). Now, the configuration of the object becomes \((8, 9)\). Now Herzberg’s regeneration protocol presented in [Herzberg] can be used to change any \((m, n)\) to \((m, n+k)\). Hence, a \((9, 11)\) Unishare split data object can be restored to a \((8, 11)\) secret data object if the fault tolerance of the \((9, 11)\) Unishare split object was higher.

### 3.2 Unishare Splitting’s fault tolerance evaluation

In this work, we are interested in finding the fault tolerance of the objects that have been split using a Unishare scheme when the failure distribution of the machines within the distributed system is known. We use \(N\) to denote the total number of machines in the distributed system and \(M\) to denote the total number of machines that have failed currently. The availability of the distributed system which is a fault tolerance metric can be stated as the fraction of data objects retrievable when \(M\) out of the \(N\) machines fail. In other words, availability can be defined as the ratio of, Number of fragment placements that makes the Unishare object retrievable even when \(M\) machines are down divided by the total number of fragment placements possible on \(N\) machines. This definition of availability assumes that the distributed system is load balanced after any new object is added which policy is usually followed in practical systems.
3.2.1 Characterizing Fault tolerance of Unishare split objects

The below is a step by step procedure (in C style) to calculate the availability of unishare objects obtained by applying unishare split on a (m, n) configuration to move to (m+k, n).

**Figure 5** – Unishare fault tolerance illustration using (9, 11) configuration

Step 1: Finding $u$, the number of imaginary primary shares plus the number of existing primary shares in the split configuration. We denote $u$ as the effective number of primary shares.

$$u = (m-1) + (n-(m-1))/(1+k)$$

Step 2: Finding $\text{den}$, the total number of ways of placing $n$ fragments on $N$ machines

a. Calculating the total number of ways of arranging $m$-1 primary shares on the total number of machines $N$

$$\text{den} = \text{den} \times \text{combi}(N,m-1)$$

b. Updating $\text{den}$ the total number of ways of arranging the sets of secondary shares on the remaining total number of machines ($N$-$m$+1)

$$\text{for}(i=0; i<=u-m; i++)$$
den=den*combi(N-(m-1)-(1+k)*i,1+k)

\textbf{Combi}(X,Y)\text{ denotes the combinatorial function representing the number of ways of picking Y balls from a total number of X balls.}

Step 3: Finding \textbf{num}, the total number of fragment placements that makes the Unishare data objects retrievable even when \textbf{M} machines have failed.

The basic idea is to find the number of ways in which at least \textbf{m} imaginary primary shares can be placed in the \textbf{N-M} servers that have not failed.

\/* For each possible value for the number of imaginary primary shares that can make the object retrievable */
for(i=m;i<=u;i++)
{
    \/* For each possible value for the number of real primary shares */
    for(j=0;j<=m-1;j++)
    {
        \/* Skip the loop if an impossible value for number of secondary shares is obtained */
        if((i-j)>u-(m-1))
            continue;

        prodterm=1;
        \/* Number of ways for placing j real primary shares in N-M of the ON servers*/
        prodterm=prodterm*combi(N-M,j);

        \/* Number of ways for placing (i-j) of the secondary share set each with 1+k shares into the remaining ON servers*/
        prodterm=prodterm*combi(u-(m-1),i-j)
        for(s=0;s<=i-j-1;s++)
            prodterm=prodterm*combi(N-M-s*(1+k)-j,1+k);

        \/* Number of ways for placing remaining m-1-j real primary shares into M of the OFF servers */
        prodterm=prodterm*combi(M,m-1-j);
/* Number of ways for placing the remaining \([u-(m-1)-(i-j)]\) secondary share set each of size \(1+k\) in \(M-j\) of the remaining OFF servers */
ON_rem=N-M-j-(i-j)*(1+k);
OFF_rem=M-(m-1)+j;
prodterm=prodterm*recsetcnt(ON_rem,OFF_rem,u,k,m,i,j,0);
num=num+prodterm;
}
}

Recursive intermediate step: Finding the number of ways for placing the remaining \([u-(m-1)-(i-j)]\) secondary share sets each of size \(1+k\) in \(M-j\) of the remaining OFF servers.

The basic idea is that a secondary share set of size \(1+k\) is part of OFF servers even if one of its \(1+k\) shares is part of the OFF servers. This is due to the \((t,t)\) split that makes the secondary share unable to be used even if 1 out of \(1+k\) within the set is placed in an OFF server.

recsetcnt(ON_rem, OFF_rem, u, k, m, i, j, setnumber)
{
    locsum=0;
    if(setnumber==u-(m-1)-(i-j))
        return 1;
    for(s=0;s<1+k;s++)
    {
        locsum=locsum+combi(ON_rem,g)*combi(OFF_rem,1+k-s)*recsetcnt(ON_rem-s,OFF_rem-(1+k-s),u,k,m,i,j,setnumber+1);
    }
    return locsum;
}

3.2.2 Unishare split’s availability for real systems

Using the availability equation derived in the previous subsection, we were able to compare the availability of Unishare objects with that of objects redistributed using Jajodia’s scheme. The availability of the \((m+k,n)\) system obtained after Jajodia
redistribution can be calculated using the standard availability measure used in [Oceanstore, Total recall, Glacier] for erasure coded fragments of data objects placed on data servers.

**Figure 6** – Availability after redistribution from (10, 20) to (11, 20) using Unishare splitting

**Figure 7** – Availability after redistribution from (11, 20) to (12, 20) using Unishare splitting
One of the drawbacks of the Unishare split method is the loss of significant number of primary shares. Since each of the \( v \) primary shares participating in the split protocol generates at least \( t \) new shares, it causes an explosion of secondary shares in comparison to the number of parent primary shares. This loss in availability can be seen from Figure 5 and 6 which show the availability of \((11, 20)\) and \((12, 20)\) configuration formed using Unishare splitting. The generation of new shares resembles a \( t \)-ary tree with secondary shares at the leaves. As a result, a significant number of replaced primary shares are lost (shares \( P10 \) and \( P11 \) as shown in figure). The inherent dependence of a large number of secondary shares on a relatively small number of primary shares affects the fault-tolerance of the resultant configuration. This is an intuitive observation verified by the analytical evaluation presented using the example in Figure 6 and Figure 7.

### 3.3 Multishare Splitting algorithm

A collaborator to this thesis [Chaitany] proposed an alternative approach that considers multiple primary shares in aggregation for splitting to avoid \( t \)-ary tree explosion.
Figure 8 shows how the secondary share $S_2$ is shared by two primary shares $P_1$ and $P_2$. This can be done by server holding $P_2$ waiting for the creation of the common share $S_2$ by using $P_1$. Then, the server holding primary share $P_2$ creates $S_3$ by combining shares $P_2$ and $S_2$ using $(2, 2)$ combining in this case. Since, the server holding the primary share waits for the creation of new shares, the completion time of a Multishare splitting is expected to be more when compared to that of a Unishare splitting scheme. The completion time of this protocol is evaluation is presented in our next section.
**Figure 9** – Creation of three secondary shares from two primary shares using Multishare splitting

Similar to having a secondary share S2 to be on the intersection of two lines as shown in Figure 9, it is possible to have different Multishare schemes creating secondary shares at the intersection of polynomials of different degrees.
Chapter 4

Implementation and evaluation of redistribution schemes

This chapter discusses the implementation and evaluation that was done to compare the completion times of Unishare and Multishare protocol. As discussed earlier, the completion time of a Multishare protocol is expected to be higher since the secondary share is being shared between the data servers. Hence, the parallelism in the splitting is lost for a Multisharing scheme when compared to a Unishare scheme and we quantify the loss using a prototype deployment in this chapter.

4.1 Implementation Details

The prototype is suited to evaluate practical distributed systems which

1. Stores data in the form of individual objects

2. Follows the principles of load balancing the storage load on each data servers after a new data object is written

In Figure 8, Metadata manager holding object id (denoted by colors), machine IP address (denoted by number 1 to 4), fragment id is implicitly denoted by the position of the machine IP address in the maintained list. Fragment id derivation for the red object: fragment 1 is in machine 1, fragment 2 is in machine 3, fragment 3 is in machine 4 and fragment 4 is in machine 5.
4.1.1 Metadata Manager

A metadata manager maintains the <Data Object id, fragment id, machine IP address> mapping. Our metadata manager implementation updates the <Data Object, fragment, machine IP address> data structure after a new object is created. The fragments of this new object are to be placed in a load balanced manner on the data servers. Hence each data server would have almost equal number of fragments after each new Data Object is written. The metadata manager also maintains the secret sharing parameters (m, n) for the individual objects.
4.1.2 Redistribution Service

A redistribution service runs at each of the data servers present in the distributed system. The request for redistribution from the metadata manager is accepted at the granularity of individual Data objects. Thus, on a Data server compromise, all the Data objects corresponding to the fragments in the compromised machine would trigger new connections to the redistribution service running at the participating servers. In the scenario shown in Figure 11, machine 2’s compromise triggers two different
redistributions (one for the white object and the other for the black object stored within a server).

**Figure 12** – Unishare redistribution from (6, 11) to (7, 11) for a data object

For the Unishare implementation shown in Figure 12, P6 is split using XOR operator into S1 and S2 at server 6. Splitting is done by creating a random block for S1 and then applying bitwise XOR of the random block S1 with P6 to create S2. After S2 is created at server holding P6, the share S2 is sent to the peer node currently holding P7. Now, the share P7 is securely deleted by the server currently holding it. The same splitting and sending a secondary share is repeated at two other servers (servers holding P8 and P10).
For the Multishare implementation shown in Figure 13, P6 is split to S1 and S2 at server 6. Then server 6 sends S2 to server 7 for storing and S2 is also sent to server 8 which forms S3 by using bitwise XOR of S2 and P8.

Bitwise XOR provides better performance benefits when compared to secret shares splitting and combining. Hence we use bitwise XOR for both Unishare and Multishare in our prototype implementations where boundary condition of \((t, t+\text{delta})\) is not reached in splitting.

![Diagram](image)

**Figure 13** – Multishare redistribution from \((6, 11)\) to \((7, 11)\) for a data object

All the message exchanges in the implementation of Unishare and Multishare redistribution used a BSD implementation of TCP/IP. The timing measurement for the protocol completion time is done at the metadata manager by counting the message sent
by the acknowledgement message sent by each data servers after splitting the fragment. The prototype uses optimal system resources by creating new child process for any computation that is required on an incoming data.

4.2 Evaluation

Our local deployment system used 20 machines within the LAN with two different configurations Intel(R) Xeon(TM) CPU 3.06GHz with physical memory 1GB RAM and cache size of 512 KB (L2 cache). All the data object sizes were maintained to be 100 KB. The experiments used network and CPU resources of these machines. We did not involve reading of data from the disk and created content from within the running services’ virtual memory.

**Figure 14** – Unishare Vs Multishare completion times for data objects of size 100KB and striped using (6, 11) secret sharing configuration
Figure 14 shows that the Unishare splitting gains in completion time due to the parallelism during the splitting phases of the protocol. Multishare protocol suffers due a larger completion time for the same reason. Since, the completion time difference between a Unishare and Multishare scheme can become very large on a typical archival system with large number of objects, a choice has to be made between the superior fault tolerance (offered by Multishare) versus superior completion time (offered by Unishare scheme).
Chapter 5

Conclusion

This thesis investigated the use of secret sharing as an encoding scheme to store long term data and presented the read/write overhead involved. Then the work focused on the problem of sustaining the security of archival data for a long period of time by increasing the threshold of the system. The solutions to this problem were designed for data being stored on heterogeneous distributed servers which have server failure patterns independent of each other. Also, the protocols were lightweight in the amount of data traffic generated and hence will be very much suitable for distributed systems on the wide area network with bandwidth constraints. The thesis introduced the differences in protocol completion time and fault tolerance that exists between different redistribution protocols. The techniques used by the work to evaluate the fault tolerance and completion time of redistribution schemes can be used by system administrators to operate the distributed storage system within specified availability and security budgets. As a future work, we plan to study practical solutions for reconfiguration in secret sharing due to node expansion & data loss.
Bibliography


[Herzberg] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or how to cope with perpetual leakage. CRYPTO ’95


[Chaitanya] Shiva Chaitanya. PhD comprehensive report, CSE Dept, Penn State University